DSP
DIGITAL SIGNAL AND IMAGE PROCESSING SERIES

Digital Signal and Image Processing using MATLAB®

Gérard Blanchet and Maurice Charbit

iSTE

Digital Signal and Image Processing using MATLAB$^{®}$

This page intentionally left blank

Digital Signal and Image Processing using MATLAB®

Gérard Blanchet
Maurice Charbit

Part of this book adapted from “Signaux et images sous Matlab : méthodes, applications et exercices corrigés” published in France by Hermès Science Publications in 2001
First published in Great Britain and the United States in 2006 by ISTE Ltd
Translated by Antoine Hervier

ISTE Ltd
6 Fitzroy Square
London W1T 5DX
UK

ISTE USA
4308 Patrice Road
Newport Beach, CA 92663
USA

www.iste.co.uk

Library of Congress Cataloging-in-Publication Data

Blanchet, Gérard.
[Signaux et images sous Matlab. English]
Digital signal and image processing using Matlab / Gérard Blanchet, Maurice Charbit.
p. cm.
Translation of: Signaux et images sous Matlab.
Includes index.
ISBN-13: 978-1-905209-13-2
ISBN-10: 1-905209-13-4
1. Signal processing--Digital techniques--Data processing. 2. MATLAB. I.Charbit, Maurice. II. Title.
TK5102.9.B545 2006
621.382'2--dc22
2006012690

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 10: 1-905209-13-4
ISBN 13: 978-1-905209-13-2

Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire.

MATLAB® is a trademark of The MathWorks, Inc. and is used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This book's use or discussion of MATLAB® software does not constitute endorsement or sponsorship by The MathWorks of a particular pedagogical approach or use of the MATLAB® software.

This page intentionally left blank

Contents

This page intentionally left blank

Preface

A practical approach through simulation

Simulation is an essential tool in any field related to engineering techniques, whether it is used for teaching purposes or in research and development.

When teaching technical subjects, lab works play an important role, as important as exercise sessions in helping students assimilate theory. The recent introduction of simulation tools has created a new way to work, halfway between exercise sessions and lab works. This is particularly the case for digital signal processing, for which the use of the MATLAB® language, or its clones, has become inevitable. Easy to learn and to use, it makes it possible to quickly illustrate a concept after introducing it in a course.

As for research and development, obtaining and displaying results often means using simulation programs based on a precise "experimental protocol", as it would be done for actual experiments in chemistry or physics.

These characteristics have led us, in a first step, to try to build a set of exercises with solutions relying for the most part on simulation; we then attempted to design an introductory course mostly based on such exercises. Although this solution cannot replace the traditional combination of lectures and lab works, we do wonder if it isn't just as effective when associated with exercise sessions and a few lectures. There is of course no end in sight to the debate on educational methods, and the amount of experiments being conducted in universities and engineering schools shows the tremendous diversity of ideas in the matter.

Basic concepts of DSP

The recent technical evolutions, along with their successions of technological feats and price drops have allowed systems based on micro-controllers and microprocessors to dominate the field of signal and image processing, at the expense of analog processing. Reduced to its simplest form, signal processing amounts to manipulating data gathered by sampling analog signals. Digital

Signal and Image Processing, or DSIP, can therefore be defined as the art of working with sequences of numbers.

The sampling theorem

The *sampling theorem* is usually the first element found in a DSIP course, because it justifies the operation by which a continuous time signal is replaced by a discrete sequence of values. It states that a signal can be perfectly reconstructed from the sequence of its samples if the *sampling frequency* is greater than a fundamental limit called the *Nyquist frequency*. If this is not the case, it results in an undesired effect called *spectrum aliasing*.

Numerical Sequences and DTFT

The *Discrete Time Fourier Transform*, or DTFT, introduced together with the sampling theorem, characterizes the spectral content of digital sequences. The analogy between the DTFT and the continuous time Fourier transform is considered, with a detailed description of its properties: linearity, translation, modulation, convolution, the Parseval relation, the Gibbs phenomenon, ripples caused by windowing, etc.

In practice, signals are only observed for a finite period of time. This "time truncation" creates ripples in the spectrum and makes it more difficult to the separate two close frequencies in the presence of noise. This leads to the concept of frequency resolution. The DTFT is a simple way of separating two frequencies, but only if the observation time is greater than the inverse of the difference between the two frequencies. The frequency resolution will allow us to introduce the reader to weighting windows. However, a more complete explanation of the concept of resolution can only be made if noise disturbing the signal is taken into account, which is why it will be studied further when random processes are considered.

The Discrete Fourier Transform, or DFT is the tool used for a numerical computation of the DTFT. Because this calculation involves a finite number of frequency values, the problem of precision has to be considered. There are a few differences in properties between the DFT and the DTFT, particularly regarding the indexing of temporal sequences that are processed modulo N. Some examples of this are the calculation of the DTFT and the DFT of a sinusoid, or the relation between discrete convolution and the DFT. At this point, the fast algorithm calculation of the DFT, also called FFT (Fast Fourier Transform), will be described in detail.

Filtering and Elements of Filter Design

Linear filtering was originally used to extract relevant signals from noise. The basic tools will be introduced: the discrete convolution, the impulse response, the frequency response, the z-transform. We will then focus on the fundamental relation between linear filtering with rational transfer functions and linear constant-coefficient recursive equations.

Filter design is described based on a few detailed examples, particularly the window method and the bilinear transform. The concepts of over-sampling and under-sampling are then introduced, some applications of which are frequency change and the reduction of quantization noise. From a broader perspective, multi-rate processing and filter banks which are described here, are two subjects that attract a lot of attention in the field of DSIP.

An introduction to images

Image processing is described in its own separate chapter. Many of the concepts used in signal processing are also used in image processing. The only difference is that two indices are used instead of one. However images have particular characteristics that require specific processing: erosion, expansion, etc. The computation time is usually much longer for images than it is for signals. It is nevertheless possible to conduct image processing with MATLAB® or one of its clones. This theme will be discussed using examples on 2D filtering, contour detection, and other types of processing in cases where the 2D nature of the images does not make them too different from a 1D signal. This chapter will also be the opportunity to discuss image compression and entropic coding.

Random Processes

Up until now, the signals used as observation models have been described by functions that depend on a finite number of well known parameters and on simple known basic functions: the sine function, the unit step function, the impulse function...This type of signal is said to be deterministic.

There are other situations where deterministic functions cannot provide us with a relevant apprehension of the variability of the phenomena. Signals must then be described by characteristics of a probabilistic nature. This requires the use of random processes, which are time-indexed sequences of random variables. Wide sense stationary processes, or WSSP, are an important category of random processes. The study of these processes is mainly based on the essential concept of *power spectral density*, or PSD. The PSD is the analog for WSSP of the square module of the Fourier transform for deterministic signals. The formulas for the linear filtering of WSSP are then laid down. Thus, we infer that WSSPs can also be described as the linear filtering of a white noise. This result leads to a large class of stationary processes: the AR process, the MA process, and the ARMA process.

Spectral Estimation

One of the main problems DSIP is concerned with is evaluating the PSD of WSSPs. In the case of continuous spectra, it can be solved by using non-parametric approaches (smooth periodograms, average periodograms, etc.) or parametric methods based on linear models (AR, MA, ARMA). As for line spectra, the most commonly used methods are the periodogram and what are called high resolution methods, which use the structures of the signal and the noise: *Prony*, *Pisarenko*, *MUSIC*, *ESPRIT*, etc.

The least squares

This chapter discusses the use of the least squares method for solving problems. This method is used in a number of problems, in fields such as spectral analysis, modelling, linear prediction, communications... We will discuss such methods as Wiener, RLS, LMS, Kalman...

Applications

This last chapter presents case studies that go a little further in depth than the examples described earlier. The emphasis is set on audio signal processing, on compression as well restoring and denoising for speech and music, and on modulation, demodulation and equalization issues for digital communications. This chapter is also an opportunity to discover typical approaches and algorithms: pitch detection, PSOLA, DTW, ACP, LBG, Viterbi...

As a Conclusion

One of the issues raised by many of those who use signal processing has to do with the artificial aspect introduced by simulation. For example, we use sampling frequencies equal to 1, and therefore frequencies with no dimension. There is a risk that the student may lose touch with the physical aspect of the phenomena and, because of that, fail to acquire the intuition of these phenomena. That is why we have tried, at least in the first chapters, to give exercises that used values with physical units: seconds, Hz, etc.

This work discusses important properties and theorems, but its objective is not to be a book on mathematics. Its only claim, and certainly an excessive one, is to show how interesting signal and image processing can be, by providing themes of study we chose because they were good examples, because they were simple, while trying not to be too trivial.

All of the subjects discussed far from cover the extent of knowledge required in this field. However they seem to us to be a solid foundation for an engineer who would happen to deal with DSIP problems.

Notations and Abbreviations

$\emptyset$		Empty Set
$\sum_{k,n}$	$=$	$\sum_k \sum_n$
$\mathrm{rect}_T(t)$	$=$	$\begin{cases} 1 & \text{when} \quad \lvert t \rvert < T/2 \\ 0 & \text{otherwise} \end{cases}$
$\mathrm{sinc}(x)$	$=$	$\dfrac{\sin(\pi x)}{\pi x}$
$\mathbb{1}(x \in A)$	$=$	$\begin{cases} 1 & \text{when } x \in A \\ 0 & \text{otherwise} \end{cases}$ (Indicator Function of A)
$(a, b]$	$=$	$\{x : a < x \leq b\}$
$\delta(t)$		$\begin{cases} \text{Dirac Distribution when } t \in \mathbb{R} \\ \text{Kronecker Symbol when } t \in \mathbb{Z} \end{cases}$
$\mathrm{Re}(z)$		Real Part of z
$\mathrm{Im}(z)$		Imaginary Part of z
i or j	$=$	$\sqrt{-1}$
$x(t) \rightleftharpoons X(f)$		Fourier Transform
$(x \star y)(t)$		Continuous Time Convolution
	$=$	$\displaystyle\int_{\mathbb{R}} x(u) y(t-u) du$
$(x \star y)(t)$		Discrete Time Convolution
	$=$	$\displaystyle\sum_{u \in \mathbb{Z}} x(u) y(t-u)$

$\mathbf{I}_N$	$(N \times N)$-dimension Identity Matrix
$\mathbf{A}^*$	Complex Conjugate of $\mathbf{A}$
$\mathbf{A}^T$	Transpose of $\mathbf{A}$
$\mathbf{A}^H$	Transpose-Conjugate of $\mathbf{A}$
$\mathbf{A}^{-1}$	Inverse Matrix of $\mathbf{A}$

$\mathbf{P}(X \in A)$	Probability that $X \in A$
$\mathbb{E}\{X\}$	Expectation Value of X
$X_c = X - \mathbb{E}\{X\}$	Zero-mean Random Variable
$\text{var}(X) = \mathbb{E}\{\lvert X_c \rvert\}^2$	Variance of X
$\mathbb{E}\{X \vert Y\}$	Conditional Expectation of X given Y

ADC	Analog to Digital Converter
ADPCM	Adaptive Differential PCM
AMI	Alternate Mark Inversion
AR	Autoregressive
ARMA	AR and MA
BER	Bit Error Rate
bps	Bits per second
cdf	Cumulative distribution function
CF	Clipping Factor
CZT	Causal z-Transform
DAC	Digital to Analog Converter
DCT	Discrete Cosine Transform
d.e./de	Difference equation
DFT	Discrete Fourier Transform
DTFT	Discrete Time Fourier Transform
DTMF	Dual Tone Multi-Frequency
dsp	Digital signal processing/processor
e.s.d./esd	Energy spectral density
FIR	Finite Impulse Response
FFT	Fast Fourier Transform
FT	Continuous Time Fourier Transform

HDB	High Density Bipolar
IDFT	Inverse Discrete Fourier Transform
i.i.d./iid	Independent and Identically Distributed
IIR	Infinite Impulse Response
ISI	InterSymbol Interference
LDA	Linear discriminant analysis
lms	Least mean squares
MA	Moving Average
MAC	Multiplication ACcumulation
OTF	Optical Transfer Function
PAM	Pulse Amplitude Modulation
PCA	Principal Component Analysis
p.d.	Probability Distribution
ppi	Points per Inch
p.s.d./PSD	Power Spectral Density
PSF	Point Spread Function
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
rls	Recursive least squares
rms	Root mean square
r.p./rp	Random process
SNR	Signal to Noise Ratio
r.v./rv	Random variable
STFT	Short Term Fourier Transform
TF	Transfer Function
WSS	Wide (Weak) Sense Stationary (Second Order) Process
ZOH	Zero-Order Hold
ZT	z-Transform

This page intentionally left blank

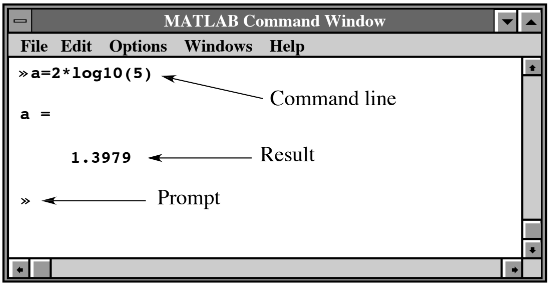
Introduction to MATLAB

In this book the name MATLAB$^{\circledR}$ (short for Matrix Laboratory) will refer to:

- the program launched by using the command `matlab` in Dos or Unix environments, or by clicking on its icon in a graphic environment such as x11, Windows, MacOS...,
- or the language defined by a vocabulary and syntax rules.

MATLAB$^{\circledR}$ is an interpreter, that is to say a program that remains in the computer's memory once it is launched. MATLAB$^{\circledR}$ displays a command window used for interpreting commands. If they are considered correct, MATLAB$^{\circledR}$ will execute them. This execution will itself lead to verifications.

Example 1 (Direct interpretation) Type `a=2*log10(5)` then <`return`>. The result is shown in a PC environment (Figure 1).

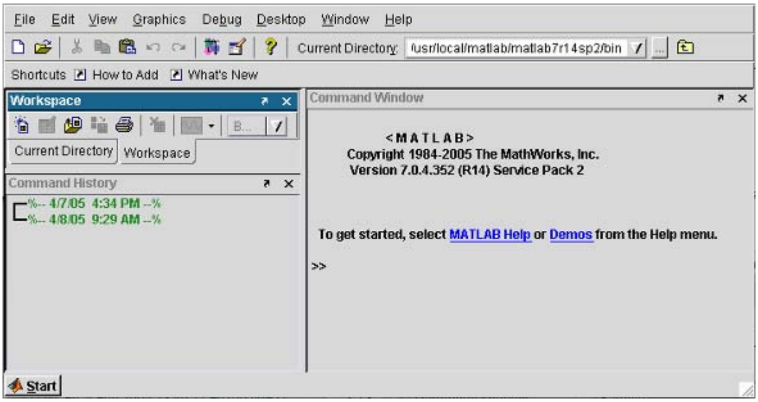


```
»a=2*log10(5)

a =

      1.3979

»
```

Figure 1 – *The* MATLAB$^{\circledR}$ *command window on MS-Windows*

Commands can be gathered together in text files called *matlab programs*. The user gives them a name that can be called from the prompt line. The MATLAB$^{\circledR}$ documentation explains how to use an editor to create such files.

This editor may either be integrated in the software or kept external (the user's favorite editor). Program files use the extension `.m`. If a program is called `prog1.m`, all the user has to do is type `prog1` in the MATLAB® command window to have it executed. MATLAB® then searches for the file in the routine directory. If it doesn't find the file there, it looks for `prog1.m` in the various files specified in the directory path. The latter can be defined directly in the command prompt window, or by using a program and executing commands such as `path`, `addpath`, `rmpath`, `genpath`, `pathtool`, `savepath` (see documentation, online help, or type `help path`).



Current Directory: /usr/local/matlab/matlab7r14sp2/bin

```
%-- 4/7/05  4:34 PM --%
%-- 4/8/05  9:29 AM --%
```

```
        <MATLAB>
 Copyright 1984-2005 The MathWorks, Inc.
 Version 7.0.4.352 (R14) Service Pack 2

To get started, select MATLAB Help or Demos from the Help menu.

>>
```

Figure 2 – *The* MATLAB® *window in an X-windows environment. The definition of the routine folder can be done directly by clicking on the icon with "..." in the top-right corner of the window. The definition of the directory path can be done by selecting the item* `set path ...` *in the menu* `file`

Clones of MATLAB® are now available. Some belong to the public domain. There also exists a compiler that allows the user to translate MATLAB® programs in machine language, making the execution quicker, and meaning that it is not required to own the interpreter.

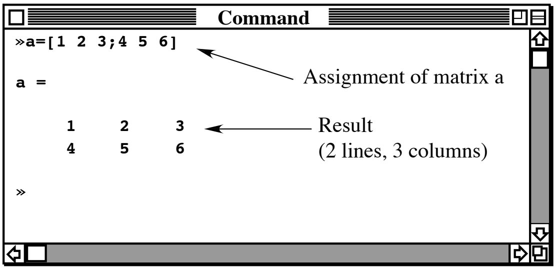
1 Variables

1.1 Vectors and matrices

The MATLAB® language is dedicated to matrix calculations and was optimized in this perspective. The variables handled as a priority are real or complex matrices. A *scalar* is a 1×1 matrix, a *column vector* is a matrix with only one column, and a *line vector* a matrix with only one line.

The notation $(\ell \times c)$ indicates that the considered variable has ℓ lines and c columns.

Example 2 (Assignment of a real matrix) Type `a=[1 2 3; 4 5 6]` at the MATLAB® prompt in the command window. The answer is shown in Figure 3.



```
»a=[1 2 3;4 5 6]

a =

     1     2     3
     4     5     6

»
```

Assignment of matrix a

Result
(2 lines, 3 columns)

Figure 3 – *Assigning a matrix*

Values are assigned to the elements of a matrix by using brackets. A space (or a comma) is a separator, and takes you to the next column, while the semicolon takes you to the next line. **Elements are indexed starting from 1**. The first index is the line number, the second one is the column number. In our example, `a(1,1)=1` and `a(2,1)=4`. The assignment `a=[1 2;3 4 5]` will of course lead to an error message, since the number of columns is different for the first and second lines.

Character strings can also be assigned to the elements of a matrix. However, the string length must be compatible with the structure of the matrix. For example, `N=['paul';'john']` would be correct, whereas `N=['paul';'peter']` would cause an error.

When the vector's components form a sequence of values separated by regular intervals, it is easier to use what is called an "implicit" loop of the type `(indD:step:indF)`. This expression refers to a list of values starting at `indD` and going up to `indF` by increments of `step`. Values cannot go beyond `indF`. The increment value `step` can be omitted if it is equal to 1.

Example 3 (Implicit enumeration) Type `a=(0:1:10)` or `a=(0:10)`. MATLAB® returns:

```
a =
      0   1   2   3   4   5   6   7   8   9   10
```

Example 4 (Incremented implicit enumeration) Type `a=(0:4:10)`. MATLAB® returns:

```
a =
      0   4   8
```

The last element of a vector is indicated by the reserved word `end`. In the previous example, `a(end)` indicates that its value is `8`.

It is possible to extend the size of a matrix. The interpreter takes care of available space by dynamically allocating memory space during the analysis of the typed phrase.

Example 5 (Extension of matrix) Type the following commands one after the other:

```
>>a=[1 2 3; 4 5 6]
a =
     1     2     3
     4     5     6
>>a=[a a]
a =
     1     2     3     1     2     3
     4     5     6     4     5     6
>>a=[1 2 3; 4 5 6];
>>a=[a;a]
a =
     1     2     3
     4     5     6
     1     2     3
     4     5     6
```

COMMENTS:

- When defining variables and objects, the language takes into account whether letters are capital or lowercase.

- Typing ";" at the end of a command line prevents the program from displaying the results of an operation.

- The display format can be modified by using the `format` command. Executing `format long`, for example, changes the number of significant digits from 5 to 15.

- The user must bear in mind that MATLAB® dedicates memory space every time a variable is used for the first time. All of the variables used during a work session are stored in the computer's memory, which means it is necessary to free space from time to time so as not to get the `OUT OF MEMORY` error message (see the `clear` command in the documentation or type `help clear`).

1.2 Arrays

Multidimensional arrays (not supported by all versions) are an extension of the normal two-dimensional matrix. One way to create such an array is to start with a 2-dimension matrix that already exists and to extend it. Type:

```
A=[1:3;4:6]
>> A
A =
     1     2     3
     4     5     6
>> A(:,:,2)=zeros(2,3), % or A(:,:,2)=0
A(:,:,1) =

     1     2     3
     4     5     6
A(:,:,2) =
     0     0     0
     0     0     0
```

The **repmat** and **cat** functions are provided in order to build multidimensional arrays.

1.3 Cells and structures

In the most recent versions of MATLAB®, there are two groups of data that are more elaborate than scalar arrays and character string arrays: the first one is called a *cell* and the second a *structure*.

In an array of *cells*, the elements can be of any nature, numerical value, character string, array, etc. Type:

```
langcell={'MATLAB',[6.5;2.3],2002}
>> langcell(2)
ans =
    [2x1 double]
>> langcell{2}
ans =
    6.5000
    2.3000
>> langcell{2}(1)
ans =
    6.5000
```

`langcell` is made up of three elements: the first one is a character string, the second one is a column vector, and the third one is a scalar. This example shows the difference in syntax between an array and a cell, a left brace ({) and a right brace (}) being used instead of a left square bracket ([) and a right square bracket (]). As for the content, `langcell(2)` refers to the vector `[6.5000;2.3]`, `langcell{2}` to the content of this vector, and `langcell{2}(1)` to the numerical value 6.5.

A *structure* is defined by the `struct` instruction. The following example defines a structure, called `langstruc`, comprising three *fields*: `Language`, `Version`, and `Year`. The instruction assigns the character string `MATLAB` to the first field, the character string `6.5` to the second field, and the numerical value 2002 to the third field:

```
>>langstruc=struct('Language','MATLAB','Version','6.5','Year',2002);
>>langstruc.Year
ans =
       2002
>>
```

The second instruction displays the content of `langstruc.Year`, which is `2002`. A `1 x 1` dimension structure is organized in the same way as an `n x 1` dimension array of cells, where `n` is the number of fields of the structure. Cells can therefore be compared to structures with unnamed fields.

The following example defines a structure named `langstruc`, comprised of two recordings. Each recording contains all three fields `Language`, `Version`, and `Year` to which were respectively assigned the sequences of two character strings `MATLAB` and `C`, of the two values 6.5 and 15.1, and of the two values 2002 and 2003:

```
>> langstruc=struct('Langage',{{'MATLAB','C'}},...
            'Version',[6.5;15.1],'Year',[2002;2003]);
>> langstruc
langstruc =
   Language: {'MATLAB'  'C'}
    Version: [2x1 double]
       Year: [2x1 double]
>> langstruc.Langage{1}
ans =
MATLAB
>> langstruc.Language(1)
ans =
    'MATLAB'
>>
```

These objects can be handled using certain functions: `isstruct`, `fieldnames`, `setfield`, `rmfield`, `cellfun`, `celldisp`, `num2cell`, `cell2mat`, `cell2struct`, `struct2cell`... An example of a conversion is as follows:

```
>> clear all
>> langcell={'MATLAB',[6.5;2.3],2002}
>> chps={'Langage','Version','Year'};
>> cell2struct(langcell,chps,2)
ans =
   Language: 'MATLAB'
    Version: 6.5000
       Year: 2002
>>
```

The 2 that is part of the instruction `cell2struct(langcell,chps,2)` indicates the dimension of `langcell` that needs to be taken into account to define the number of fields. Here, for example, `size(langcell,2)` means that the number of fields is 3.

2 Operations and functions

2.1 Matrix operations

The main matrix operations are the following:

- The $(+, \times)$ operations, sum and multiplication of two matrices.

  Example 6 (Multiplication of matrices) Type the following commands:

  ```
  >>a=[1 2; 3 4] * [5;6]
  a =
      17
      39
  >>size(a)
  ans =
       2     1
  ```

  The command `size(a)` returns “1 2” giving us the number of lines and the number of columns of `a`.

- The backslash provides the solution to the linear problem $\mathbf{A}\mathbf{x} = \mathbf{b}$ in the form `x=A\b`. If $\mathbf{A}$ is a full-rank square matrix, this amounts to multiplying on the left by the inverse matrix. Otherwise, the solution is given *in the least squares* sense[1].

  Example 7 (Solving a linear system) Type:

  ```
  >>A=[1 2;2 3]; b=[1;1]; % Full rank square matrix
  >>x=inv(A)*b            % Solution using the inverse
  x =
      -1
       1
  >>x=A\b     % Solution using the system resolution
  x =
      -1
       1
  ```

- The operation `A/B` amounts to performing the operation `B'\A'`.

- The operation `^` carries out the exponentiation of the argument, which can be a fractional scalar, positive or negative, or a matrix.

- The apostrophe `'` is used for the *transpose-conjugate* or *transconjugate*. As a reminder, if the $(N \times N)$ matrix $\mathbf{A}$ is the conjugate-transpose of $\mathbf{B}$, then $\mathbf{A} = \mathbf{B}^H$ and we have $[a_{ij}] = [b_{ji}^*]$ for $1 \leq i, j \leq N$.

[1] The problem of solving a linear system in the least-squares sense plays a crucial role in signal processing. This will be discussed further in Chapter 11.

Example 8 (A few operations) Type the following commands:

```
>>a=[2 0;1 3];
>>a^2
ans =
     4     0
     5     9
>>a^.5
ans =
    1.4142         0
    0.3178    1.7321
>>a=(0:3);
>>b=(0:3);
>>c=b*a'
c =
    14
>>d=b'*a
d =
     0     0     0     0
     0     1     2     3
     0     2     4     6
     0     3     6     9
```

The vectors a and b are real (4×1) line vectors. The scalar c is therefore equal to the scalar product of the vectors a and b. On the other hand, d is a "multiplication table"-type (4×4) matrix.

2.2 Pointwise operations

The operations ".×", "./" and ".^" work *term by term*. The phrase *pointwise operations* is also used. For example, if $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ are two matrices of the *same dimension*, `A .* B` returns the matrix $[a_{ij}b_{ij}]$.

Example 9 (Pointwise operations) Type the following commands and check the result:

```
>>clear  % Free data memory space
>>a=(1:3)' * (1:4); b=(5:7)' * (1:2:7);
>>c = a.* b;
>>d=a ./ b;
>>e = a .^(.5);
>>a,b,c,d,e
```

In this sequence of instructions:

- a and b are two matrices with 3 lines and 4 columns. They are obtained by multiplying a dimension 3 column vector and a dimension 4 line vector;

- c is a matrix that has $c_i = a_i \times b_i$ as its generic element;

- d is a matrix whose element $d_i = a_i/b_i$;

- e is a matrix whose element $e_i = \sqrt{a_i}$.

Example 10 (Alternating sequence)
`(-1).^[0:9]` leads to a sequence of alternating 1 and -1.

COMMENTS:

- in term by term operations, matrices must have the same dimensions;
- while the operation `'` (apostrophe) transconjugates a matrix, the operation `.'` (period-apostrophe) transposes without conjugating.

Example 11 (Transposition and transconjugation) Type:

```
>>a=[1+j 2;3 4]
a =
   1.0000 + 1.0000i   2.0000
   3.0000             4.0000
>>a'
ans =
   1.0000 - 1.0000i   3.0000
   2.0000             4.0000
>>a.'
ans =
   1.0000 + 1.0000i   3.0000
   2.0000             4.0000
```

2.3 Constants and initialization

The constants `pi`, `i`, `j` are predefined: `pi=3.14159265358979`$\cdots$, `i` = `j` = $\sqrt{-1}$. Keep in mind that executing the instruction `pi=4` makes `pi` lose its predefined value. It is recommended not to use `pi`, `i` and `j` as variables in a program.

`eps`, `realmin` and `realmax` are other constants provided for limit test purposes. Their values are respectively: $2.220446049250313e-16$, $2.225073858507201e-308$ and $1.797693134862316e+308$.

2.4 Predefined matrices

The following commands are used to obtain certain particular matrices:

- `ones(L,C)` returns a matrix with L lines and C columns containing nothing but ones. `ones(1,N)` returns a line vector made up of N ones;
- `zeros(L,C)` returns a matrix with L lines and C columns containing nothing but zeros;
- `eye(N)` returns the $N \times N$ identity matrix (ones on the diagonal and zeroes everywhere else);

- `eye(L,C)` returns the $N \times N$ identity matrix $\mathbf{I}_N$, where N refers to the smaller of the two numbers L and C, completed by a matrix containing nothing but zeros, so as to obtain a matrix with L lines and C columns. `eye(1,N)`, for example, will return a line vector with one "1" followed by $N-1$ "0";
- `randn(L,C)` returns a matrix with L lines and C columns containing a centered gaussian distributed sample with a variance equal to 1;
- `rand(L,C)` returns a matrix with L lines and C columns containing a sample uniformly distributed on the interval $(0,1)$;
- aside from the usual matrices, such as Hilbert, Hadamard, Vandermonde, etc., a large number of predefined matrices are available using the `gallery` function. For a list of these matrices, type `help gallery`.

The `reshape` function is used to change the size of a matrix, for example, to go from a (2×6) matrix to a (3×4) matrix (refer to documentation, or type `help`). This change of size can also be done directly, as shown in the following example:

```
a=[(1:6);(7:12)];  % 2*6 matrix
c=zeros(3,4);      % Predimensioning
c(:)=a;            % Column by column filling-out
```

which would be the equivalent of `c=reshape(a,3,4)`. The `zeros(3,4)` command initializes the choice of size for the matrix `c`. The purpose of the next instruction `c(:)=a` is to fill out the matrix `c`, `column by column`, with the sequence of 12 values taken from `a column by column`. `a` and `c` must have the *exact same* number of elements.

Example 12 (Predefined matrices) The instructions:

```
x=[ones(1,5);-ones(1,5)];
y=zeros(1,10); y(:)=x
```

return a line vector containing 10 alternate 1 and -1. As we have seen, the same thing can be done with `(-1).^[0:9]`.

2.5 Mathematical functions

Certain functions handle matrices only as an array of values. This is the case for functions such as: `abs`, `sqrt`, `exp`, `cos`, `sin`, `log`, `tan`, `acos`, `asin`, `atan`, etc.

Example 13 (Exponential function) Type:

```
T=1024; tims=(0:T-1);
%===== Three frequencies
fq =[.01 .013 .014];
%===== Complex Signal
sig = exp(2*j*pi*tims'*fq);
%===== Displaying of the real part of the complex exponential
%      for fq=0.01 that is to say cos(2*pi*0.01*n)
plot(tims, real(sig(:,1)))
```

`tims` is a (1×1024) *line* vector and therefore `tims'*fq` is a (1024×3) matrix. You can see this for yourself by typing, at the end of the previous program, the command `whos`:

```
>>whos
    Name          Size          Elements      Bytes   Density   Complex
       T        1 by 1                 1          8      Full        No
      fq        1 by 3                 3         24      Full        No
     sig     1024 by 3              3072      49152      Full       Yes
    tims        1 by 1024           1024       8192      Full        No
```

The instruction `sig = exp(2*j*pi*tims'*fq);` applies the exponential function to each of the elements of the matrix `2*j*pi*tims'*fq`. The result is the (1024×3) matrix `sig`. In the last instruction `plot(tims,real(sig(:,1)))`, `sig(:,1)` refers to the last column of the matrix `sig`.

Built-in functions

There is a large number of library functions that can be called using the MATLAB® language. Some are provided with the MATLAB® interpreter, while others have to be paid for, as part of extra modules.

Among the functions available in the basic version, the user will for example find mathematical functions such as the exponential `exp`, the logarithm `log`, the usual trigonometric functions... or functions that have more to do with signals and images such as the Fourier transform `fft`, the 2D convolution `conv2`, etc.

Some of these functions are written in the MATLAB® language, while others are written in machine language, for reasons of execution speed.

Example 14 ("Source programs" and compiled programs) Type: `type compan`. MATLAB® displays the text of the `compan` function, which can be found in one of the folders of your hard-drive. However, if the instruction `type fft` is executed, MATLAB® returns:

```
??? Built-in function
```

meaning that this function is compiled and that its source code cannot be accessed.

In the most recent versions of MATLAB®, many functions that used to be written as "`.m`" programs (see paragraph 6) were rewritten and now appear as "`Built-in`".

2.6 Matrix functions

As we have seen, the `exp(a)` command calculates the exponential of each element of the matrix `a`. This operation must not be confused with the matrix exponential. The letter "m" at the end of the functions `expm(A)`, `logm(A)`, `sqrtm(A)` indicates that we are dealing with matrix functions. For example, $e^{\mathbf{A}}$ is defined by:

$$e^{\mathbf{A}} = \mathbf{I} + \frac{\mathbf{A}}{1!} + \cdots + \frac{\mathbf{A}^k}{k!} + \cdots$$

and is obtained with the function `expm(A)`.

There is also a function called `funm` that can be used to calculate any function of a matrix. Type `help funm`.

2.7 Other useful functions

The `eig` function returns the eigenvalues and the eigenvectors of a matrix. The `poly` function returns the characteristic polynomial associated to a matrix, or a polynomial whose roots are a given vector. The `roots` function returns the roots of a polynomial.

Example 15 (A few functions - 1) Type:

```
>>a=[1 1 1];
>>rr=roots(a)
rr =
  -0.5000 + 0.8660i
  -0.5000 - 0.8660i
>>poly(rr)
ans =
    1.0000    1.0000    1.0000
```

The values of the complex roots of the polynomial $a(x) = x^2 + x + 1$ are obtained with `roots(a)`.

Example 16 (A few functions - 2) Type:

```
>>a=[1 2;1 1];
>>poly(a)
ans =
    1.0000   -2.0000   -1.0000
>>[vp,md]=eig(a)
vp =
    0.8165   -0.8165
    0.5774    0.5774
md =
    2.4142         0
         0   -0.4142
>>roots(poly(a))
ans =
    2.4142
   -0.4142
```

In this example, **a** is a (2×2) matrix, its characteristic polynomial `poly(a)` is equal to $\det(\lambda\mathbf{I} - \mathbf{a}) = \lambda^2 - 2\lambda - 1$. The eigenvectors of **a** are given by `vp`. `md` is the diagonal matrix bearing the eigenvalues on its diagonal, which are also the roots of the characteristic polynomial.

2.8 Logical operators on boolean variables

The logical operators AND (symbol `&`), OR (symbol `|`), and NOT (symbol `~`) operate on boolean quantities. The "false" boolean value is coded as 0 and "true" as a non-zero value. Boolean quantities can be used in structures such as "`if ... elseif ... else ... end`", "`switch ... case ... otherwise ... end`" or "`while ... end`".

Example 17 (Logical functions) Type:

```
>>x=1;
>>if x==0,
A=[1 2];
else
A=[2 1];
end
>>A
A =
     2     1
```

`isnan`, `isinf`, `isfinite`, `isstr`, `ischar`, etc. are boolean functions used for testing purposes.

2.9 Program loops

The `for ... end` program structure works as a calculation loop.

Example 18 (Program loops) Type:

```
>>A=[1 .5; .5 .25];
>>M=eye(2,2); % Unit Matrix
>>for k=1:5
M = M * A;    % Calculation of the consecutive powers of A
end
```

The loop can be written in a single line with `for k=1:5; M = M * A; end`.

As is the case for many interpreters, loops tend to deteriorate calculation performances considerably. The user is therefore advised not to use them, by replacing them with matrix functions when possible.

Example 19 (Avoiding loops) Type:

```
>>a=randn(400);
>>for k=1:400
  for m=1:400
  b(k,m)=a(k,m)^2;
  end
  end
>>c=a .^2;
```

The last instruction returns a matrix `c` identical to the matrix `b`. However its execution is much faster.

3 Graphically displaying results

Display windows are chosen using the command `figure(n)`, where `n` is a window number. Inside the active window, the `plot` command can be used to graphically display results:

- If `x` and `y` are two real vectors of the same length, the `plot(x,y)` displays the graph of `y` as a function of `x`.
- If `x` and `y` are two real *matrices* of the same size, the `plot(x,y)` command displays the first column of `y` as a function of the first column of `x`, the second column of `y` as a function of the second column of `x`, and so on until there are no columns left. Each line has its own color.
- If `x` is a real vector with a length of N, and `y` is a size $(N \times K)$ real matrix, the `plot(x,y)` command displays the K graphs corresponding to the K columns of `y` as a function of `x`.
- If `x` is a complex vector, the `plot(x)` displays the graph of the imaginary part of x as a function of the real part of `x` (see example 20).
- If the command `subplot(3,2,4)` or `subplot(324)` is added before the `plot` command, the graph is divided in six "sub-windows" organized in three lines of two columns each, and the display is done in sub-window number 4.

Example 20 (Drawing of a circle)
Type the following program:

```
clear;
z=exp(2*pi*j*[0:100]/100);
figure(1); plot(z); axis('square');
figure(2); subplot(121); plot(z); axis([-1.2 1.2 -1.2 1.2]);
subplot(122); plot(z); axis('square');
```

The `axis('square')` command forces the display to appear in a square. The second `axis` command forces particular values on the minima and the maxima of the x- and y-coordinates. The third one makes the calculation of the minima and maxima automatic.

The `zoom` command is used to zoom in on a particular part of the graph.

In the recent versions of MATLAB®, windows and graphs are objects whose properties can be consulted and modified using `get` and `set`. These properties can also be accessed from the pull-down menus, meaning that the user does not have to reprogram them. See exercise 2.8, page 80 which describes another way to go about this.

Example 21 (Drawing an ellipse)
The exponents T and H refer to the transposition and the transposition-conjugation respectively. If the matrix is real, then of course $\mathbf{Y}^H = \mathbf{Y}^T$. In its matrix form, the equation of an ellipse is:

$$(\mathbf{X} - \mathbf{X}_0)^H \mathbf{E}(\mathbf{X} - \mathbf{X}_0) = c$$

where c is a positive constant, $\mathbf{X}_0$ is a dimension 2 vector characterizing the center of the ellipse, and $\mathbf{E}$ is a 2×2 positive matrix, meaning that, for any complex vector $\mathbf{Y}$, $\mathbf{Y}^H\mathbf{E}\mathbf{Y}$ is a positive number. A simple way of obtaining a positive matrix is to take any real matrix $\mathbf{G}$ and to calculate $\mathbf{G}^T\mathbf{G}$.

By diagonalizing $\mathbf{E}$, we get $\mathbf{E} = \mathbf{P}\mathbf{D}\mathbf{P}^H$ where $\mathbf{D}$ is a diagonal matrix with all its diagonal elements positive, and $\mathbf{P}$ a unitary matrix, that is to say such that $\mathbf{P}\mathbf{P}^H = \mathbf{P}^H\mathbf{P} = \mathbf{I}$ where $\mathbf{I}$ refers the identity matrix.

Let us assume $\mathbf{F} = \mathbf{P}\mathbf{D}^{1/2}\mathbf{P}^H$. Incidentally, we have $\mathbf{F} = \mathbf{F}^H$ and $\mathbf{F}^H\mathbf{F} = \mathbf{E}$. $\mathbf{F}$ is called the square root of $\mathbf{E}$. Starting with this, we have $(\mathbf{X}-\mathbf{X}_0)^H\mathbf{E}(\mathbf{X}-\mathbf{X}_0) = (\mathbf{X}-\mathbf{X}_0)^H\mathbf{F}^H\mathbf{F}(\mathbf{X}-\mathbf{X}_0)$. By assuming $\mathbf{Y} = \mathbf{F}(\mathbf{X}-\mathbf{X}_0)$, we get $\mathbf{Y}^H\mathbf{Y} = c$ which is the equation of a circle of center O and radius $\sqrt{c}$ in a set of orthonormal coordinates. This leads us to a calculation procedure of N points of the ellipse characterized by $\mathbf{X}_0$, $\mathbf{E}$ and c:

1. Calculate $\mathbf{Y} = \sqrt{c}[\cos\theta \;\; \sin\theta]$ for θ from 0 to 2π in steps of $2\pi/N$.
2. Make the variable change $\mathbf{X} = \mathbf{X}_0 + \mathbf{F}^{-1}\mathbf{Y}$.

Starting with this procedure, we now write a function with the ellipse's center defined by any vector `X0`, the positive matrix `E`, and the constant `c` as input parameters.

HINT: type:

```
function ellipse(X0, E, c)
%%===================================================%
%% Drawing an ellipse                                %
%% SYNOPSIS: ELLIPSE(X0, E, c)                       %
%%    X0 = Coordinates of the ellipse's center (2x1) %
%%    E  = A positive (2x2) matrix                   %
%%    c  = Scale Factor                              %
%%===================================================%
N=100; theta = (0:N) * (2*pi) ./ N ;
Y = sqrt(c)*[cos(theta);sin(theta)];
Fm1=inv(sqrtm(E));
X = diag(X0)*ones(2,N+1)+Fm1*Y;
plot(X(1,:),X(2,:)); set(gca,'DataAspectRatio',[1 1 1])
return
```

Test this function[2] with the following program, choosing several values of c and:

$$\mathbf{X}_0 = [0\ 0] \text{ and } \mathbf{E} = \begin{bmatrix} 1.3628 & 0.7566 \\ 0.7566 & 0.5166 \end{bmatrix}$$

```
>>X0=[0 0];
>>E=[1.3628 .7566;.7566 .5166];
>>c=1;
>>ellipse(X0, E,c)
```

We displayed in Figure 4 the results obtained for $c = \{1, 2, 3, 4, 5\}$. ■

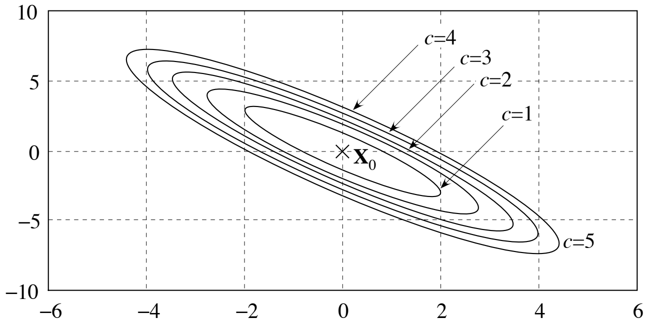


Figure 4 – *Drawing of several ellipses*

[2] Save this function under the name `ellipse.m`. It will be used later on.

4 Converting numbers to character strings

As an example, let us consider the `text(x,y,'text')` command. It allows the user to add text to a graph, placed at coordinates (x, y). To add a numerical value to the `text` command, it must first be converted to a character string. This can be done with the `num2str` command.

Example 22 (Numbers and character strings) Type:

```
fe=10;
valfe=num2str(fe)
```

The `sprintf` command can also be used to build a character string. In fact, it is used by `num2str`.

Example 23 (Creating a character string) Type:

```
fq=[10.5 20.566];
valf=sprintf('F1 = %+15.2f, F2 = %4.2e', fq(1),fq(2))
```

The expression `sprintf(...)` leads to a character string obtained by converting the numerical value to the format specified by `format`. For example, the `%10.4f` format converts the given value with 4 decimal points. For more information, it is recommended to read the `printf` function's description in C language.

The functions `str2num` and `hex2num` should also be looked into.

5 Input/output

MATLAB® makes it possible to perform input-output operations from the keyboard, on the screen (as it was explained in the previous paragraph with `sprintf`) or on files. Here are the main functions:

- `input, ginput, ...` for keyboard acquisition;
- `disp, sprintf, ...` to display on the screen;
- `gtext, plot, grid, title, ...` to display in a graph;
- `load, save` to load or save parts of the variables in a file, or all of them, in a format specific to MATLAB®. By default, files have the extension `.mat`.

  For versions newer than MATLAB-4, it is still possible to ensure that files are compatible with version 4 by using the option `-v4` of the `save` command;
- `fopen, fread, fwrite` for input-output with formatting.

Example 24 (Input/output in a file) Type:

```
clear; x=[1:100];
fid=fopen('try1.dat','w');
fwrite(fid,x,'short');      % Writing
fclose(fid)
%==========
fid=fopen('try1.dat','r');
y=fread(fid,'short')        % Reading
fclose(fid);
```

This program creates the `try1.dat` file of 16 bit integers, then reads its content in variable `y`.

6 Program writing

We have seen that several commands can be grouped together in a file that can be run by typing its name in the command window. Such a program often uses functions with names corresponding to the `.m` files that contain them, and not the ones declared by the reserved word **`function`**!

It is however possible to gather several functions in a single `m` file, so long as these functions are called from inside the file, but they are not visible from the outside.

The first lines of comment (lines starting with a % symbol) of a program (or a function) are displayed when the **`help`** command is executed followed by the name of the program. It is strongly advised to systematically use this possibility to write down a synthetic description of the functions used.

MATLAB® authorizes the use of procedures written in an evolved language such as C, Pascal or Fortran. These programs belong to the type called MEX. Throughout the rest of this book, we will use only predefined functions and those we are going to build in the MATLAB® language.

MATLAB® also makes it possible to create programs with a graphic interface, combining buttons, pull-down menus, scrolling windows, etc. Using these possibilities is a good way of building "press-a-button" demonstrations or lab works that help to emphasize certain properties. The demonstrations included with MATLAB® are an excellent source of documentation for creating such programs.

Part I

Deterministic Signals

This page intentionally left blank

Chapter 1

Signal Fundamentals

Although this work is mainly focused on discrete-time signals, a discussion of continuous-time signals cannot be avoided, for at least two reasons:

- The first reason is that the quantities we will be using – taken from numeric sequences – are taken from continuous-time signal sampling. What is meant is that the numeric value of a signal, such as speech, or an electroencephalogram reading, etc., is measured at regular intervals.

- The second reason is that for some developments, we will have to use mathematical tools such as *Fourier series* or *Fourier transforms* of continuous-time signals.

The objective is not an extensive display of the knowledge needed in the field of deterministic signal processing. Many other books have already done that quite well. We will merely give the main definitions and properties useful to further developments. We will also take the opportunity to mention *systems* in a somewhat restricted meaning, this word referring to what are called *filters*.

1.1 The concept of signal

A *deterministic* continuous-time signal is defined as a function of the real *time* variable t:

$$\text{Signal} = \text{function } x(t),\, t \in \mathbb{R}$$

The space made up of these functions is completed by the *Dirac pulse* distribution, or $\delta(t)$ function.

The following functions spaces are considered:

- $L_1(\mathbb{R})$ is the vector space of summable functions such that $\int_{\mathbb{R}} |x(t)|dt < +\infty$;

- $L_1(a,b)$ is the vector space (vector sub-space of $L_1(\mathbb{R})$) of functions such that $\int_a^b |x(t)|dt < +\infty$;
- $L_2(\mathbb{R})$ is the vector space of *finite energy functions* such that $\int_{\mathbb{R}} |x(t)|^2 dt < +\infty$;
- $L_2(a,b)$ is the vector space (vector sub-space of $L_2(\mathbb{R})$) of functions such that $\int_a^b |x(t)|^2 dt < +\infty$;
- the set of "finite power" functions characterized by:

$$\lim_{T\to+\infty} \frac{1}{T}\int_{-T/2}^{T/2} |x(t)|^2 dt < +\infty$$

$L_2(0,T)$ has the structure of what is called a *Hilbert space* structure with respect to the scalar product $\int x(t)y^{(t)}dt$, a property that is often used for decomposing functions, for example in the case of *Fourier series*.

In the course of our work, we will need to deal with a particular type of signal, in sets that have already been defined, taken from $\mathbb{R}^+$.

Definition 1.1 (Causal and anticausal signals) *Signals $x(t)$ such that $x(t) = 0$ for $t < 0$ are said to be causal. Signals $x(t)$ such that $x(t) = 0$ for $t \geq 0$ are said to be anticausal.*

1.1.1 A few signals

We will often be using particular functions characteristic of typical behaviors. Here are some important examples:

- the *unit step* function or *Heaviside function* is defined by:

$$u(t) = \mathbb{1}(t \in (0, +\infty[) \tag{1.1}$$

  Its value at the origin, $t = 0$, is arbitrary. Most of the time, it is chosen equal to 1/2. The unit step can be used to show causality: $x(t)$ is causal if $x(t) = x(t)u(t)$;

- the *sign* function is defined using the unit step by $\text{sign}(t) = 2u(t) - 1$;
- the *gate* or *rectangle* function is defined by:

$$\text{rect}_T(t) = \mathbb{1}(t \in (-T/2, T/2)) = u(t + T/2) - u(t - T/2) \tag{1.2}$$

  It will be used to express the fact that a signal is observed over a finite time horizon, with a duration of T. The phrases *rectangular windowing* and *rectangular truncation* of $x(t)$ are also used: $x_T(t) = x(t)\text{rect}_T(t)$;

- the *pulse*, or *Dirac function*, has the following properties which serve the purpose of calculation rules:

  1. $\int_{\mathbb{R}} \delta(t)dt = 1$ and $\int_{\mathbb{R}} \delta(t)x(t)dt = x(0)$.
  2. $x(t) = \int_{\mathbb{R}} x(u)\delta(t-u)du = (x \star \delta)(t)$ ($\star$ is the *convolution operation*).
  3. $x(t)\delta(t - t_0) = x(t_0)\delta(t - t_0)$.
  4. $(x(u) \star \delta(u - t_0))(t) = (x \star \delta)(t - t_0) = x(t - t_0)$.
  5. $\delta(at) = \delta(t)/|a|$ for $a \neq 0$.
  6. $\forall t$, $\int_{-\infty}^{t} \delta(u)du = \mathbb{1}(t \in (0, +\infty)) = u(t)$ and therefore $du(t)/dt = \delta(t)$. This result makes it possible to define the derivative of a function with a jump discontinuity at a time t_0. Let $x(t) = x_0(t) + au(t - t_0)$ where $x_0(t)$ is assumed to be differentiable. We have $dx(t)/dt = dx_0(t)/dt + a\delta(t - t_0)$;

- the *sine function* is defined by:

$$x(t) = x_0 \sin(\Omega_0 t + \phi) = x_0 \sin(2\pi F_0 t + \phi) \tag{1.3}$$

  x_0 is the peak amplitude of the signal, Ω_0 its angular frequency (in radians/s), ϕ its phase at the origin, $F_0 = \Omega_0/2\pi$ its frequency (in Hz) and $T = 1/F_0$ its period;

- the *complex exponential function* is defined by:

$$x(t) = x_0 \exp(2j\pi F_0 t + j\phi) \tag{1.4}$$

- the *sine cardinal* is defined by $\text{sinc}(t) = \sin(\pi t)/\pi t$. It is equal to 0 for all integers except $t = 0$ (hence its name). We have $\int_{\mathbb{R}} \text{sinc}(t)dt = 1$, $\int_{\mathbb{R}} \text{sinc}(u)\text{sinc}(u-t)du = \text{sinc}(t)$ and the following orthogonality property, for $n \in \mathbb{N}$:

$$\int_{\mathbb{R}} \text{sinc}(u)\text{sinc}(u - n)du = \begin{cases} 1 & \text{with} \quad n = 0 \\ 0 & \text{with} \quad n \neq 0 \end{cases}$$

1.1.2 Spectral representation of signals

Fourier series

A periodic signal with a period of $T = 1/F_0$ may be decomposed as a sum of complex exponentials, a sum we will refer to as *Fourier series*[1]:

$$\begin{cases} x(t) \overset{F.S.}{=} \displaystyle\sum_{k=-\infty}^{+\infty} X_k e^{2j\pi k F_0 t} \\ X_k = \displaystyle\frac{1}{T}\int_0^T x(t) e^{-2j\pi k F_0 t} dt \end{cases} \tag{1.5}$$

$F_0 = 1/T$ is called *fundamental* frequency, and its multiples are called *harmonic* frequencies. A few comments should be made:

- a signal with a bounded support on (t_1, t_2) is also expandable in a Fourier series, but the series converges to the periodized function outside of the (t_1, t_2) interval;
- expression 1.5 indicates that X_k is the k-th component of $x(t)$ in the orthonormal basis of the complex exponentials $\{T^{-1/2}e^{2j\pi k F_0 t}\}_{k\in\mathbb{Z}}$ in the *Hilbert space* $L_2(0, T)$;
- $x_M(t) = \sum_{k=-M}^{M} X_k e^{2j\pi k t/T}$ is the best length M approximation of $x(t)$ in the sense of the least squares;
- when $x(t)$ is *continuous*, $x_M(t)$ converges uniformly to $x(t)$ for any t, when $M \to +\infty$;
- if $x(t)$ shows first order discontinuities, $x_M(t)$ converges in quadratic mean, but not uniformly, to $x(t)$. This is indicated by the symbol $\overset{F.S.}{=}$ in the developed expression. At discontinuity points, $x_M(t)$ will converge to the half-sum of the left and right limits of $x(t)$. Finally, $x_M(t)$ can show some non-evanescent oscillations in the neighborhoods of all discontinuities. This phenomenon is referred to as the *Gibbs phenomenon*;
- we have *Parseval's relation*:

$$\frac{1}{T}\int_0^T |x(t)|^2 dt = \sum_{k\in\mathbb{Z}} |X_k|^2 = \int_{\mathbb{R}} \sum_{k\in\mathbb{Z}} |X_k|^2 \delta(f - kF_0) df \tag{1.6}$$

  where $\delta(f - kF_0)$ refers to the Dirac distribution at point kF_0. Because the first member of 1.6 is by definition the signal's power, the sequence $\{|X_k|^2\}$ can be interpreted as the power distribution along the frequency axis. It is also called *power spectral density*, or *PSD*.

[1] We will only be using the complex exponential decomposition, since it easily leads to the one with the sine and cosine functions.

Fourier transform

The spectral contents $X(f)$ of the function $x(t) \in L_1(\mathbb{R}) \cap L_2(\mathbb{R})$ can be represented by an integral that uses complex exponentials, an integral we will call *Fourier transform*:

$$X(f) = \int_{\mathbb{R}} x(t)e^{-2j\pi f t}dt \quad \longleftrightarrow \quad x(t) = \int_{\mathbb{R}} X(f)e^{2j\pi f t}df \tag{1.7}$$

$|X(f)|$ is called *spectrum* of $x(t)$. The Fourier transform's main properties are summarized in Appendix A1.

The convolution property 14.1 leads to *Parseval's formula*:

$$\int_{\mathbb{R}} |x(t)|^2 dt = \int_{\mathbb{R}} |X(f)|^2 df \tag{1.8}$$

Because the left member of 1.8 is, by definition, the signal's energy, $|X(f)|^2$ can be interpreted as the energy distribution along the frequency axis. It is also called *energy spectral density*, or *esd*.

More generally, we have:

$$\int_{\mathbb{R}} x(t)y^*(t)dt = \int_{\mathbb{R}} X(f)Y^*(f)df \tag{1.9}$$

Example 1.1 (Analytical signal)
Let $x(t)$ be a time-continuous real signal. The *analytical signal* associated with $x(t)$ is the signal $z(t)$ that has $Z(f) = 2U(f)X(f)$ as its Fourier transform, where $X(f)$ is the Fourier transform of $x(t)$ and $U(f)$ is the unit step function equal to 1 if $f > 0$ and 0 if $f < 0$. $U(0)$ is chosen equal to $1/2$.

Using the properties of the continuous-time Fourier transform, show that the real part of $z(t)$ is equal to $x(t)$, and determine its imaginary part called the *Hilbert transform* of $x(t)$.

HINT: let:

$$p(t) = \mathrm{Re}(z(t)) = (z(t) + z^*(t))/2$$

Using the Fourier transforms, we get:

$$P(f) = (Z(f) + Z^*(-f))/2 = U(f)X(f) + U(-f)X^*(-f)$$

Because $x(t)$ is real, $X(f) = X^*(-f)$, and therefore, $P(f) = X(f)$, which means $p(t) = x(t)$. As a conclusion, $\mathrm{Re}(z(t)) = x(t)$.

Likewise, let:

$$q(t) = \mathrm{Im}(z(t)) = (z(t) - z^*(t))/2j$$

Using the Fourier transforms, we get:

$$\begin{aligned} Q(f) &= (Z(f) - Z^*(-f))/2j = -j(U(f)X(f) - U(-f)X^*(-f)) \\ &= -j(U(f) - U(-f))X(f) \end{aligned}$$

Because $U(f)-U(-f)$ is the sign(f) function, $Q(f) = -j\text{sign}(f)X(f)$. This equation can be interpreted as filtering (see paragraph 1.2) with the complex gain filter $-j\text{sign}(f)$. Its gain is equal to 1, meaning that the Fourier transforms of the output and input have the same modulus, $|Q(f)| = |X(f)|$.

As a conclusion, the analytical signal associated with the real signal $x(t)$ is written:

$$z(t) = x(t) + j\hat{x}(t)$$

where $\hat{x}(t)$ refers to the Hilbert transform of $x(t)$. ■

1.2 The Concept of system

A *system* transforms the signal $x(t)$ and delivers a signal $y(t)$, the result of this alteration. We will refer to this transformation as $y(t) = \mathcal{T}[x(u), t]$, and $x(t)$ and $y(t)$ will be called the input and the output of the system respectively.

Filters

A *filter* with $x(t)$ as the input and $y(t)$ as the output is a system defined by:

$$y(t) = \int_{\mathbb{R}} x(u)h(t-u)du = \int_{\mathbb{R}} x(t-u)h(u)du \tag{1.10}$$

The existence of the integral has to do with how the set $\mathcal{X}$ of considered signals $x(t)$ is chosen. Among the sets that have practical interest, two of them play a fundamental role: the signals that have a Fourier transform and those made up of a linear mix of complex exponentials.

Certain conditions have to be met:

– first, in the case of $\mathcal{X}$ sets that show some practical interest, such a system is *linear*: $\mathcal{T}[a_1x_1(u) + a_2x_2(u), t] = a_1\mathcal{T}[x_1(u), t] + a_2\mathcal{T}[x_2(u), t]$;

– second, it is *time-invariant*: $\mathcal{T}[ax(u), t - t_0] = \mathcal{T}[ax(u - t_0), t]$. Another way of expressing it is to say that the output is independent of the time origin.

Example 1.2 (Counterexample) The system defined by:

$$y(t) = \int_0^t x(u)du$$

is linear but is time-dependent.

HINT: the output corresponding to the signal $x(t - t_0)$:

$$\tilde{y}(t) = \int_0^t x(u-t_0)du = \int_{-t_0}^{t-t_0} x(v)dv$$

is different from:

$$y(t-t_0) = \int_0^{t-t_0} x(u)du$$

which is the output at time $t - t_0$ when $x(t)$ is used as the input signal. ■

Impulse response

The $h(t)$ function found in 1.10 is called the filter's *impulse response.* The output $y(t)$, convolution product of $x(t)$ and $h(t)$, is denoted $y(t) = (x \star h)(t)$.

A causal system is a system that depends only on the current and previous inputs. This means that a filter is causal if $h(t) = 0$ for $t < 0$.

Frequency response

Let us first consider the case of $x(t)$ signals that have a Fourier transform $X(f)$. Using the convolution product's property leads us to:

$$Y(f) = X(f)H(f)$$

The $H(f)$ function is called the filter's *frequency response* or *complex gain.*

Let us now take a look at signals $x(t)$ that are a linear mix of complex exponentials. Because of the linearity property, all we have to do is calculate the output with $x(t) = \exp(2j\pi F_0 t)$ as the input. We get:

$$y(t) = \int_{\mathbb{R}} \exp(2j\pi F_0(t-u))h(u)du = H(F_0)\exp(2j\pi F_0 t)$$

Therefore, the complex output signal $H(F_0)\exp(2j\pi F_0 t)$ corresponds to the complex exponential $\exp(2j\pi F_0 t)$. In this case, complex exponentials are called the *eigenfunctions* of the filters (the eigenvalue beeing $H(F_0)$).

Stability

A system is said to be *BIBO stable* if for any Bounded Input the Output is Bounded. Stability is an essential system property.

A filter is BIBO stable if and only if:

$$\int_{\mathbb{R}} |h(u)|du < +\infty$$

1.3 Summary

The following table contains some definitions and properties that will be used throughout the next lessons.

Continuous time	Discrete time
Fourier transform $X(f) = \int_{\mathbb{R}} x(t)e^{-2j\pi ft}dt$ $x(t) = \int_{\mathbb{R}} X(f)e^{2j\pi ft}df$	Discrete time Fourier transform $X(f) = \sum_{n\in\mathbb{Z}} x(n)e^{-2j\pi nf}$ $x(n) = \int_{-1/2}^{1/2} X(f)e^{2j\pi nf}df$
Linear filter $(t \in \mathbb{R})$ $(x \star h)(t) \leftrightarrow X(f)H(f)$ BIBO stability $\Leftrightarrow \int_{\mathbb{R}} \lvert h(t)\rvert dt < +\infty$	Linear filter $(n \in \mathbb{Z})$ $(x \star h)(n) \leftrightarrow X(f)H(f)$ BIBO stability $\Leftrightarrow \sum_{n\in\mathbb{Z}} \lvert h(n)\rvert < +\infty$
Fourier series $X(k) = \frac{1}{T}\int_0^T x(t)e^{-2j\pi kt/T}dt$ $x(t) = \sum_{k\in\mathbb{Z}} X(k)e^{2j\pi kt/T}$	Discrete Fourier transform $X(k) = \sum_{n=0}^{N-1} x(n)e^{-2j\pi kn/N}$ $x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{2j\pi nk/N}$
Bilateral Laplace transform $X(s) = \int_{\mathbb{R}} x(t)e^{-st}dt$ $x(t) = \frac{1}{2j\pi}\int_{C-j\infty}^{C+j\infty} X(s)e^{st}ds$	z-Transform $X(z) = \sum_{n\in\mathbb{Z}} x(n)z^{-n}$ $x(n) = \frac{1}{2j\pi}\oint_{\Gamma} X(z)z^{n-1}dz$
Filter $(t \in \mathbb{R})$ $(x \star h)(t) \leftrightarrow X(s)H(s)$ BIBO stability $\Leftrightarrow$ imaginary axis $\subset$ domain of convergence of $H(s)$.	Filter $(n \in \mathbb{Z})$ $(x \star h)(n) \leftrightarrow X(z)H(z)$ BIBO stability $\Leftrightarrow$ unit circle $\subset$ domain of convergence of $H(z)$.

Chapter 2

Discrete Time Signals and Sampling

Signal processing consists of handling data in order to extract information considered relevant, or to modify them so as to give them useful properties: extracting, for example, information on a plane's speed or distance from a RADAR signal, making an old and decayed sound recording clearer, synthesizing a sentence on an answering machine, transmitting information through a communication channel, etc.

The processing is called *digital* if it deals with a discrete sequence of values $\{x_1, x_2 \ldots\}$. There are two types of scenarios: either the observation is already a sequence of numbers, as is the case for example for economic data, either the observed phenomenon is "continuous-time", and the signal's value $x(t)$ must then be measured at regular intervals.

This second scenario has tremendous practical applications. This is why an entire paragraph of this chapter is devoted to the operation called *sampling.*

The acquisition chain is described in Figure 2.1.

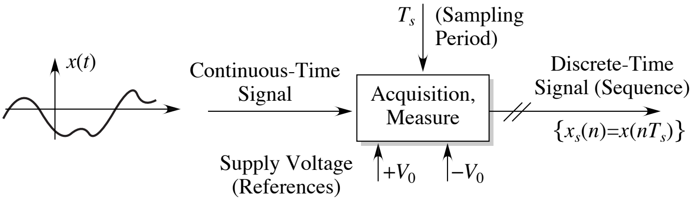


Figure 2.1 – *Digital signal acquisition*

The essential part of the acquisition device is usually the *analog-to-digital converter*, or *ADC*, which samples the value of the input voltage at regular intervals – every T_s seconds – and provides a coded representation at the output.

To be absolutely correct, this coded value is not exactly equal to the value of $x(nT_s)$. However, in the course of this chapter, we will assume that $x_s(n) = x(nT_s)$. The sequence of these numerical values will be referred to as the *digital* *signal*, or more plainly as the *signal.*

T_s is called the *sampling period* and $F_s = 1/T_s$ the *sampling frequency.* We will discuss later the problems caused by the gap between the actual value and the coded value, which is called *quantization noise.*

Obviously, the sampling frequency must be high enough "in order not to lose too much information" – a concept we will discuss later on – from the original signal, and there is a connection between this frequency and the sampled signal's "frequential content". Anybody who conducts experiments knows this "graph plotting principle": when the signal's value changes quickly (presence of high frequencies), "many" points have to be plotted (it would actually be preferable to use the phrase high point density), whereas when the signal's value changes slowly (presence of low frequencies), less points need to be plotted.

To sum up, the signal sampling must be done in such a way that the numerical sequence $\{x_s(n)\}$ alone is enough to reconstruct the continuous-time signal. The sampling theorem specifies the conditions that need to be met for perfect reconstruction to be possible.

2.1 The sampling theorem

Let $x(t)$ be a continuous signal, with $X(F)$ its Fourier transform, which will also be called the *spectrum.* The sample sequence measured at the frequency $F_s = 1/T_s$ is denoted by $x_s(n) = x(nT_s)$.

Definition 2.1 *When $X(F) \neq 0$ for $F \in (B_1, B_2)$ and $X(F) = 0$ everywhere else, $x(t)$ is said to be (B_1, B_2) band-limited. If $x(t)$ is real, its Fourier transform has a property called hermitian symmetry, meaning that $X(F) = X^*(-F)$, and the frequency band's expression is $(-B, +B)$. A common misuse of language consists of referring to the signal as a B-band signal.*

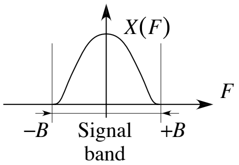


Figure 2.2 – *$(-B, +B)$ Band-limited real signal*

2.1.1 Perfect reconstruction

Our goal is to reconstruct $x(t)$, at every time t, using the sampling sequence $x_s(n) = x(nT_s)$, while imposing a "reconstruction scheme" defined by the expression (2.1):

$$y(t) = \sum_{n=-\infty}^{+\infty} x(nT_s)h(t - nT_s) \tag{2.1}$$

where $h(t)$ is called a *reconstruction function.* Notice that 2.1 is linear with respect to $x(nT_s)$. In order to reach this objective, two questions have to be answered:

1. Is there a class of signals $x(t)$ large enough for $y(t)$ to be identical to $x(t)$?
2. If that is the case, what is the expression of $h(t)$?

The answers to these questions are provided by the sampling theorem 2.1.

Theorem 2.1 (Sampling theorem)
Let $x(t)$ be a (B_1, B_2) band-limited signal, real or complex, and let $\{x(nT_s)\}$ be its sample sequence, then there are two possible cases:

1. *If $F_s = 1/T_s$ is such that $F_s \geq B_2 - B_1$, then $x(t)$ can be perfectly reconstructed from its samples $x(nT_s)$ using the expression:*

$$x(t) = \sum_{n=-\infty}^{+\infty} x(nT_s)h_{(B_1,B_2)}(t - nT_s) \tag{2.2}$$

where the FT of the reconstruction function $h_{(B_1,B_2)}(t)$ is:

$$H_{(B_1,B_2)}(F) = \frac{1}{F_s}\ \mathbb{1}(F \in (B_1, B_2)) \tag{2.3}$$

2. *If $F_s = 1/T_s < B_2 - B_1$, perfect reconstruction turns out to be impossible because of the "spectrum aliasing" phenomenon.*

The proof uses the *Poisson summation formula* which gives the relation between $X(F)$ and the values of $x(t)$ at sampling times nT_s, and makes it possible to determine the expression of the spectrum of the signal $y(t)$ defined by equation 2.1.

Lemma 2.1 (Poisson formula) *Let $x(t)$ be a signal, and $X(F)$ its Fourier transform. Then for any T_s:*

$$\frac{1}{T_s} \sum_{k=-\infty}^{+\infty} X(F - kF_s) = \sum_{n=-\infty}^{+\infty} x(nT_s)\exp(-2j\pi nFT_s) \tag{2.4}$$

where the left member is assumed to be a continuous function of F.

HINT: the left member of equation 2.4 will be written $\sigma(F)$. By construction, $\sigma(F)$ is periodic with period $F_s = 1/T_s$. Therefore, $\sigma(F)$ can be expanded in a Fourier series that can be expressed as $\sigma(F) \overset{S.F.}{=} \sum_{n=-\infty}^{+\infty} c_n e^{-2j\pi nF/F_s}$ where:

$$\begin{aligned}
c_n &= \frac{1}{F_s}\int_0^{F_s} \sigma(F) e^{2j\pi nF/F_s} dF \\
&= \int_0^{F_s} e^{2j\pi nFT_s}\left[\sum_{k=-\infty}^{+\infty} X(F-kF_s)\right] dF \\
&= \sum_{k=-\infty}^{+\infty}\left[\int_0^{F_s} e^{2j\pi nFT_s} X(F-kF_s) dF\right] \\
&= \sum_{k=-\infty}^{+\infty}\left[\int_{-kF_s}^{-kF_s+F_s} e^{2j\pi nuT_s} X(u) du\right] = \int_{-\infty}^{+\infty} e^{2j\pi kuT_s} X(u) du
\end{aligned}$$

where we have assumed that $u = F - kF_s$ in order to go from the first-to-last line to the last line. By referring to property 1.7, which gives us the inverse Fourier transform, we infer that $c_n = x(nT_s)$, thus demonstrating formula 2.4. ■

We did not go into the detail of all the hypotheses necessary to justify the previous calculations. We will assume that these calculations are valid. Other books written on the Fourier transform can be looked up for a more rigorous approach.

We will use the following definition for the *discrete-time Fourier transform*. We will see another completely equivalent expression of it (definition 2.4, expression 2.21), but more frequently used in the case of numerical sequences.

Definition 2.2 (DTFT) *The sum* $\sum_{n=-\infty}^{+\infty} x(nT_s)\exp(-2j\pi nFT_s)$ *is called the Discrete-Time Fourier Transform (DTFT) of the sequence* $\{x(nT_s)\}$*.*

We now go back to the sampling theorem. By using the fact that the Fourier transform of $h(t-nT_s)$ is $H(F)e^{-2j\pi nFT_s}$, the Fourier transform of $y(t)$, defined by 2.1, can be written:

$$\begin{aligned}
Y(F) &= \sum_{n=-\infty}^{+\infty} x(nT_s)\times H(F)e^{-2j\pi nFT_s} = H(F)\sum_{n=-\infty}^{+\infty} x(nT_s)e^{-2j\pi nFT_s} \\
&= \frac{H(F)}{T_s}\sum_{k=-\infty}^{+\infty} X(F-kF_s) \qquad (2.5)
\end{aligned}$$

Therefore, if $F_s \geq B_2 - B_1$, the different contributions $X(F-kF_s)$ do not overlap, and by simply assuming $H_{(B_1,B_2)}(F) = T_s\mathbb{1}(F \in (B_1, B_2))$, $Y(F)$

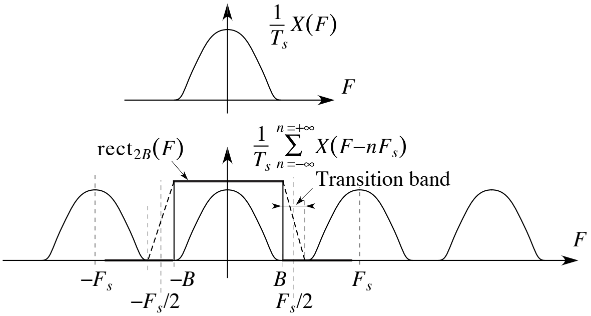


Figure 2.3 – *Real signal reconstruction*

coincides exactly with $X(F)$. Figure 2.3 illustrates this case for a real signal. In this case, $B_1 = -B$ and $B_2 = B$.

Except if specified otherwise, we will assume from now on that $x(t)$ is real. The *sufficient* reconstruction condition can be written as follows:

$$F_s \geq 2B \tag{2.6}$$

The limit frequency $2B$ is called the *Nyquist frequency*. Still in the same case, the Fourier transform of a possible reconstruction function is $H_B(F) = T_s\text{rect}_{2B}(F)$, and therefore:

$$h_B(t) = \frac{\sin(2\pi Bt)}{\pi F_s t} \tag{2.7}$$

It should be noted that the filter $H_B(F) = T_s\text{rect}_{2B}(F)$ is not the only possible filter. If F_s is assumed to be strictly greater than $2B$, then we can choose a filter with larger transitions bands (see Figure 2.3), making it easier to design.

When there is no possible doubt, we will not indicate the dependence on B, and simply write $h(t)$ instead of $h_B(t)$.

Anti-aliasing filter

The reconstruction formula 2.1, is, according to the Poisson's formula 2.4, associated with the periodization of the spectrum $X(F)$ with the period F_s. It follows that, for $F_s < 2B$, the different non-zero parts of the spectrum overlap, making perfect reconstruction impossible. The overlapping phenomenon is called *spectrum aliasing.*

Figure 2.4 illustrates the spectrum aliasing phenomenon for a real signal whose frequential content is of the “low-pass” type, implicitly meaning that it “fills up” the band $(-F_s/2, +F_s/2)$.

Except in some particular cases (see example 2.1 and modulations), we will assume that spectrum signals are of this type, or that they can be modified to fit this description.

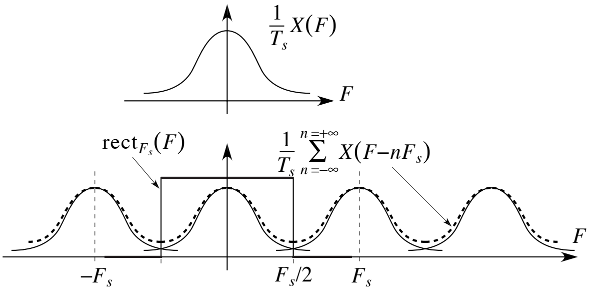


Figure 2.4 – *The aliasing phenomenon*

For a real signal, showing aliasing means that the frequencies beyond the frequency $F_s/2$ can be "brought back" to the $(-F_s/2, +F_s/2)$ band.

In practice, the following cases will occur:

1. The sampling frequency is imposed: if, knowing how the data is used, the aliasing phenomenon is considered to "cause damage", the appropriate procedure for sampling a real signal requires the use of low-pass filtering called *anti-aliasing filtering* which eliminates the components of the frequencies higher than $F_s/2$.

2. The sampling frequency is not imposed: in this case, it can be chosen high enough so that the aliased components of the signal do not alter the expected results. If this is not possible, F_s is set, and the situation becomes the same as in the first case.

Speech signals are a good example. If they are sampled at 8,000 Hz, an extremely common value, high enough to make the person speaking recognizable and understandable, and if no anti-aliasing filtering is done, the reconstructed signal contains a "hissing" noise. This alone justifies the use of an anti-aliasing filter. The irretrievable loss of high frequency components is actually better than the presence of aliasing.

Figure 2.5 illustrates the case of a "low-pass", prefiltered, real signal to prevent aliasing.

In general, it is important to understand that anti-aliasing filtering must be done in the band that is considered essential (*useful band*) to the unaliased signal reconstruction. The low-pass filtering mentioned here corresponds to a low-pass sampled signal.

The following general rule can be stated:

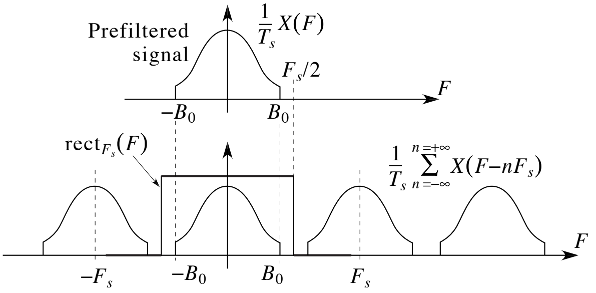


Figure 2.5 – *Absence of aliasing after* $[-B_0, +B_0]$ *filtering* $[-B_0, +B_0]$

The sampling operation of a signal at the frequency F_s *must be preceded by an anti-aliasing filtering with a gain equal to* 1 *and with a width of* F_s *in the useful band.*

The following example illustrates the case of a real band-pass signal, therefore ($\mathcal{B} = [-F_{\max}, -F_{\min}] \cup [F_{\min}, F_{\max}]$) band-limited. If this was not the case, an anti-aliasing filtering in the useful band $\mathcal{B}$ would be necessary.

Example 2.1 (Sampling of a narrowband signal)
Let $x(t)$ be a $(F_{\min}, F_{\max})$ "band-limited" real signal. If $(F_{\max} + F_{\min})/2 \gg (F_{\max} - F_{\min})$, the signal is called a *narrowband* signal. Determine the sampling frequencies that allow the perfect reconstruction of $x(t)$.

HINT: the application of formula 2.6 leads to $F_s > 2F_{\max}$. We will now show that it is still possible to conduct a slower sampling. In order to do this, let us consider the Fourier transform of the signal $y(t) = \sum_{n=-\infty}^{+\infty} x(nT_s)h(t-nT_s)$ given by expression 2.5:

$$Y(F) = \frac{H(F)}{T_s} \sum_{k=-\infty}^{+\infty} X(F - kF_s)$$

This leads us to the conclusion that, in order for $X(F)$ to coincide with $Y(F)$, the two following conditions have to be met:

1. the periodized function $\sum X(F - kF_s)$ shows no aliasing (Figure 2.6);
2. the function $H(F) = T_s$ for $F_{\min} < |F| < F_{\max}$ and 0 otherwise (see Figure 2.6).

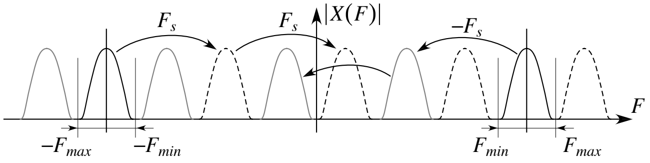


Figure 2.6 – *Narrowband signal* $(F_{\min}, F_{\max})$

The non-aliasing condition (1) is met for frequencies for which both the following inequalities are true:

$$kF_s - F_{\min} < F_{\min} \quad \text{and} \quad (k+1)F_s - F_{\max} > F_{\max}$$

which is equivalent to the condition:

$$\frac{2F_{\max}}{k+1} < F_s < \frac{2F_{\min}}{k} \tag{2.8}$$

where k is an integer such that $k \leq k_0$ where k_0 is the integer part of $F_{\min}/(F_{\max} - F_{\min})$. For $k = 0$, We encounter once again the Nyquist frequency $2F_{\max}$, but if $k_0 > 0$, we get possible sampling frequencies that are smaller than $2F_{\max}$. Condition (2) leads to the following reconstruction function:

$$h(t) = T_s \frac{\sin(\pi \Delta F t)}{\pi t} \cos(2\pi F_0 t)$$

with $\Delta F = F_{\max} - F_{\min}$ and $F_0 = (F_{\max} + F_{\min})/2$. ■

Causal approximation of the reconstruction formula

In order to calculate $x(t)$ at time t, expression 2.2 requires that all the future samples beyond t (absence of causality) and up until infinity be known. However, because $h(t)$ decreases like $1/t$, it is possible to approximate $x(t)$ by using a finite number of samples before and after t. A delay is therefore necessary for practical reconstruction. For $t \in (mT_s, (m+1)T_s)$ and a high enough value of L, this can be written as follows:

$$x(t) \approx \sum_{k=m-L}^{m+L} x(kT_s)h(t - kT_s) \tag{2.9}$$

Of course, this expression only allows the calculation of $x(t)$, in the interval $(mT_s, (m+1)T_s)$, if $x((m+L)T_s)$ is known. Reconstruction can therefore be accomplished by tolerating a delay LT_s.

We will see in Chapter 4 an implementation based on the insertion of zeros followed by a filtering. Polynomial interpolations are other methods which can be used.

Spectrum aliasing and ambiguity

Let us consider the continuous-time sine signal:

$$x(t) = \cos(2\pi F_0 t) \text{ with } F_0 = 350 \text{ Hz} \tag{2.10}$$

sampled at a frequency of $F_s = 800$ Hz. The sample sequence can be written as follows:

$$x_s(n) = x(n/F_s) = \cos(2\pi f_0 n) \text{ with } f_0 = F_0/F_s$$

Let us also consider the continuous-time sine signal of frequency $F_1 = F_s - F_0$:

$$y(t) = \cos(2\pi F_1 t) \text{ with } F_1 = 1{,}150 \text{ Hz}$$

sampled at the same frequency $F_s = 800$ Hz. The sample sequence is:

$$y_s(n) = y(n/F_s) = \cos(2\pi f_1 n) \text{ with } f_1 = F_1/F_s$$

Using $F_1 = F_0 + F_s$, we get $f_1 = f_0 + 1$. Replacing in $y(n)$ leads us to:

$$y_s(n) = \cos(2\pi(f_0 + 1)n) = \cos(2\pi f_0 n) = x_s(n)$$

This result shows that the use of samples taken at a frequency of F_s alone is not enough to be able to tell signal $x(t)$ from $y(t)$. Therefore reconstruction will lead to the same signals, whether it is done from samples $x_s(n)$ or $y_s(n)$. In the case of the signal $x(t)$, the result is accurate, but it is false for $y(t)$: we started with a frequency of $F_1 = 1{,}150$ Hz and ended up with a signal frequency of $F_0 = 350$ Hz.

For a given signal, for any integer k, it is not possible to distinguish F_0 from $F_1 = F_0 + kF_s$, $k \in \mathit{ZintSet}$, which is called the *image frequency* of F_0 relative to F_s. This is the *ambiguity* due to the spectrum aliasing phenomenon (or generally speaking to the Poisson formula).

Example 2.2 (Ambiguity)
In the previous example, we now consider $F_1 = 450$ Hz. Write a program illustrating this case. The continuous-time signal will be visualized over a period of 5 ms, as well as the samples $x_s(n)$ and $y_s(n)$.

HINT: type the program:

```
%===== ALIASEXPLE.M
Fs=800; Te=1/Fs; F0=350; F1=Fs-F0;
tmax=.005; mtm=[0:tmax/100:tmax];
xt=cos(2*pi*F0*mtm);
yt=cos(2*pi*F1*mtm);
plot(mtm,[xt' yt']); grid; hold on;
n1max=floor(tmax/Te);
```

```
mtm=[0:n1max]*Te; % Sampling times
xen0=cos(2*pi*F0*mtm);
plot(mtm,xen0,'o');
xen1=cos(2*pi*F1*mtm);
plot(mtm,xen1,'x'); hold off
```

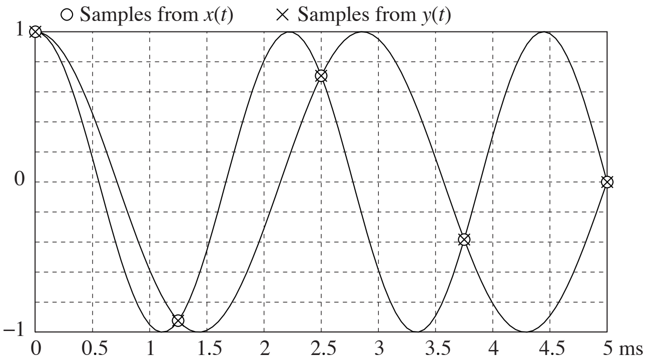


Figure 2.7 – *An illustration of aliasing*

We obtain the same result $y_s(n) = x_s(n)$. In the case of $x(t) = \sin(2\pi F_0 t)$ we obtain $y_s(n) = -x_s(n)$. ■

Listen to the spectrum aliasing

We will now perform two simple experiments that will allow us to "hear" the spectrum aliasing phenomenon.

The first one simply consists of recording speech at a frequency of 8 kHz, then to take one out of every two samples, and to listen to the signal obtained at a frequency of 4 kHz. Type the following program:

```
%===== SPEECHALIAS.M
%===== .mat file containing the speech signal
load speechsig
Fs=8000; N=length(x); xr=x(1:2:N);
soundsc(x,Fs); pause; soundsc(xr,Fs/2)
```

"Hissing" noises can be heard in the restored signal. We will come back to this example in exercise 4.15 and give the proper method for undersampling a signal while avoiding aliasing.

In the second example, we create a digital signal from the sampling of a signal defined by its continuous-time expression. Instead of working the way the sampling theorem tells us to, we are going to cause spectrum aliasing.

Consider the continuous-time signal given by 2.11:

$$x(t) = A\cos(\Theta(t)) \text{ with } t \in \mathbb{R} \tag{2.11}$$

The time dependent function defined by:

$$F_i(t) = \frac{1}{2\pi}\frac{d\Theta}{dt} \tag{2.12}$$

is called the *instantaneous frequency.* If $x(t)$ is a F_0 frequency sine signal, the instantaneous frequency is equal to F_0. In general, $x(t)$ is said to be *frequency modulated.* Unfortunately, there is no simple expression for the spectrum of $x(t)$. However we can suspect that, for the most part, the energy can be located in the frequency band scanned by the function $F_i(t)$.

Consider, for example, the case of an instantaneous frequency that varies linearly with time, which can be written:

$$F_i(t) = F_0 + \lambda t$$

where λ is expressed in Hz/s. By observing the signal over long enough periods of time, between instants 0 and T, the frequency should vary linearly between F_0 and $F_1 = F_0 + \lambda T$. We will now determine the expression of $x(t)$. By integrating $F_i(t)$, and by assuming that $F_i(0) = 0$, we get:

$$\Theta(t) = 2\pi F_0 t + \pi\lambda t^2$$

The following program creates the samples taken at a frequency of $F_s =$ 8,000 Hz of the signal $x(t)$, for a period of $T = 2$ s, with $F_0 = 1{,}000$ Hz and for a value of λ that we will change, so as to sweep frequency ranges of varying widths:

```
%===== MODULFREQ.M
lambda=1000;      % Parameter (1000 ou 2000)
Fs=8000;          % Sampling Freq.
F0=1000;          % Initialization Freq.
T=2;              % Observation time
it=(0:Fs*T-1)/Fs; % Time Vector
theta=2*pi*F0*it+pi*lambda*(it .^2);
x=cos(theta);
soundsc(x,Fs)     % Result
```

The `soundsc(x,Fe)` function reconstructs a continuous-time signal from samples `x` at a sampling frequency `Fs`, and sends it to the calculator's audio output.

First listen to the signal obtained for $\lambda = 1{,}000$ Hz/s, as it has been defined in the example. You can hear a sound going from a low-pitched frequency to a high-pitched frequency, because the instantaneous frequency varies linearly from $F_0 = 1{,}000$ Hz to $F_1 = 3{,}000$ Hz.

Now listen to a signal for $\lambda = 2{,}000$ Hz/s. This time, a low-pitched sound can be heard, "rising" to a higher frequency, and finally going back down to a low frequency. This result is rather unexpected, since the instantaneous frequency varies linearly from $F_0 = 1{,}000$ Hz to $F_1 = 5{,}000$ Hz. This is simply the consequence of the spectrum aliasing phenomenon. Because the sampling frequency is equal to 8,000 Hz, the frequencies beyond $F_s/2 = 4{,}000$ Hz are aliased in the (0 Hz - 4,000 Hz) band. This means that, during reconstruction (see paragraph 2.1.2), when the instantaneous frequency varies between 4,000 Hz and 5,000 Hz, the `soundsc(x,Fe)` function sees the signal as a frequency varying from 4,000 Hz to 3,000 Hz.

Interpolation and visual impressions

As we are now going to see, a sampling frequency equal to or slightly greater than the Nyquist frequency leads to a continuous-time signal that cannot be clearly identified simply by looking at it. This means that the eye, or more precisely the brain, is a rather poor interpolator.

To observe this effect, consider a sine function with a frequency of 80 Hz and a first sampling frequency of $F_{s1} = 200$ samples per second. This sampling frequency is greater than the Nyquist frequency, equal only to 160 Hz, and therefore is high enough to reconstruct the sine function. Now consider the same signal sampled at a frequency of $F_{s2} = 1{,}500$ samples per second. The following program creates and plots the sequences of values corresponding to these two sampling frequencies, over a period of 60ms:

```
%===== SINUS80.M
f0=80;                   % Sinus Freq.
obsdur=0.06;             % Observation Time
Fs1=200; Fs2=1500;       % Sampling Freq.
n1=round(obsdur*Fs1); n2=round(obsdur*Fs2);
tps1=[0:n1-1]/Fs1; tps2=[0:n2-1]/Fs2;
s1=3*sin(2*pi*f0*tps1); s2=3*sin(2*pi*f0*tps2);
subplot(211); plot(tps1,s1,'x'); grid
subplot(212); plot(tps2,s2,'x'); grid
```

The resulting plot is shown on Figure 2.8.

As you can see, the continuous-time sine function is not recognizable from the top figure, corresponding to the 200 Hz sampling. On the other hand, the bottom figure, corresponding to the 1,500 Hz sampling, gives a very good visual impression of a sine function.

It should be pointed out that if the sampling frequency is chosen to be much greater than the number of pixels on the screen, the dots on the graph are displayed as an almost "continuous-time" trajectory. An interpolation function can then be used to build the trajectory. In Chapter 5, we will create an interpolation program (exercise 4.14) that calculates $(R-1)$ intermediate points

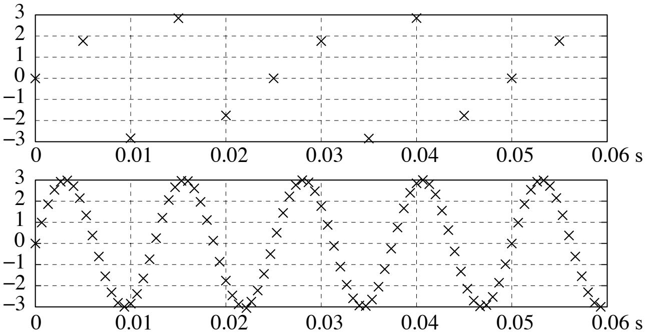


Figure 2.8 – *Drawing of the 80 Hz sine function, sampled at a frequency of 200 Hz (top graph) and at a frequency of 1,500 Hz (bottom graph)*

regularly spaced out between each point of a sequence. R is called the interpolation order. In the following exercises, the function used to plot continuous-time signals may be used, simply by imposing $R \gg 1$.

Exercise 2.1 (An illustration of the sampling theorem)
Consider the function $x(t) = \sin(2\pi F_0 t)$, sampled at a frequency of F_s.

1. What signal results from perfect reconstruction for $F_0 = 200$ Hz and $F_s = 500$ Hz?

2. A 200 Hz sine function is sampled at a frequency of $F_s = 250$ Hz. What signal is obtained by using the ideal formula for perfect reconstruction?

3. Write a program:

   – displaying a 200 Hz sine function,

   – displaying 10 of its samples taken at the frequency F_s,

   – displaying the reconstructed signal (expression 2.2). The reconstruction will be performed using the `filter` function in the following way:

   ```
   xti = filter(hn,1,xtr)
   ```

   where `hn` is the sample sequence $h(nT_s)$ of $h(t)$ (expression 2.7) and `xtr` the sample sequence of the sine function completed with zeros,

   – and checking the accuracy of the results for questions 1 and 2.

2.1.2 Digital-to-analog conversion

Reconstructing a continuous-time analog signal from a numerical sequence is done by using a *Digital-to-Analog Converter*, or *DAC*. The DAC blocks the value of $x(nT_s)$ during the time interval $(nT_s, (n+1)T_s)$ where $T_s = 1/F_s$. The converter is called a *Zero-Order Hold* (*ZOH*).

$x_0(t)$, the ZOH's output signal, is shaped like a "staircase". Its expression is:

$$x_0(t) = \sum_n x(nT_s)h_0(t - nT_s) = \sum_n x(nT_s)\mathbb{1}(t \in (nT_s, nT_s + T_s))$$

Compared to the original signal $x(t)$, the signal $x_0(t)$ has some of its power in high frequencies due to the presence of steep transitions. The frequential study (figure 2.9) clearly shows this behavior: the Poisson formula 2.4 gives us the following expression for the Fourier transform of $x_0(t)$:

$$X_0(F) = H_0(F) \sum_{n=-\infty}^{+\infty} X(F - n/T_s)$$

with:

$$H_0(F) = \frac{\sin(\pi F T_s)}{\pi F T_s} e^{-j\pi F T_s} = \mathrm{sinc}(FT_s)e^{-j\pi F T_s}$$

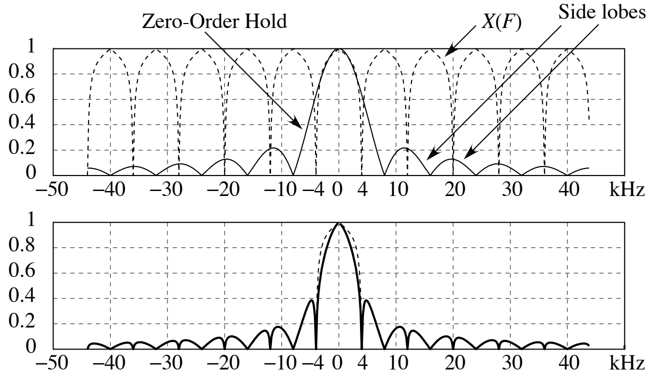


Figure 2.9 – *Spectrum modulus at the ZOH's output*

The shape of $|X_0(F)|$ shows two kinds of distortion when comparing the original signal $x(t)$ with the reconstructed signal $x_0(t)$:

1. The first one is due to the presence of the term $|\mathrm{sinc}(FT_s)|$ which deforms the original spectrum in the band $(-F_s/2, F_s/2)$.

2. The second one has to do with the spectrum's periodization and the presence of side lobes for the function $|\text{sinc}(fT_s)|$ beyond $F_s/2$, and particularly in the band $(F_s/2, F_s)$ corresponding to the first side lobe.

   For example, in the case of an "audio application" sampled at a frequency of $F_s = 8{,}000$ Hz, these components appear between 4,000 Hz and 8,000 Hz and are perfectly audible. One possible solution is to apply a low-pass filter to the ZOH's output.

In general, the greater the sampling frequency (compared to the band of the signal $x(t)$), the weaker these distortions will be. This is why for some devices, the ZOH is preceded by an *interpolation operation.* This processing technique is explained on page 154.

2.2 Plotting a signal as a function of time

The sampling theorem makes it possible to go from a continuous-time signal to a sequence of values obtained by using a filter with a gain equal to 1 in the band $(-F_s/2, F_s/2)$, followed by a sampling procedure at a frequency of F_s. From now on, and except if specified otherwise, we will only be considering discrete-time signals, that is to say sequences of values, that we will study plotted as functions of time and frequency. These two kinds of plotting, which are equivalent by definition, are nevertheless both useful when interpreting the phenomena we are dealing with.

Digital signals

The first model, called the *temporal* model, for a digital signal, is made up of the values of its samples. As is the case for continuous-time, the support of these sequences can be limited to $\mathbb{N}^+$.

Definition 2.3 (Causal and anticausal signals) *The causal signals $x(n)$ are such that $x(n) = 0$ for $n < 0$. If all the elements of the sequence are equal to zero for $n \geq 0$, the sequence is said to be anticausal.*

In the same way, some "basic" signals have to be considered to come up with an ideal model for certain of the observed signals. This is the case for example for a sine voltage or for very short pulses used to characterize the behaviour of certain "systems". Here is an (incomplete) list of some of these signals:

- The *unit pulse* defined by:

$$\delta(n) = \begin{cases} 1 & \text{for } n = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

- The *unit step* defined by:

$$u(n) = \begin{cases} 1 & \text{for } n \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

- The *sign function* defined by:

$$\text{sign}(n) = 2u(n) - 1 = \begin{cases} +1 & \text{for} \quad n \geq 0 \\ -1 & \text{for} \quad n < 0 \end{cases} \tag{2.15}$$

- The *gate function* or *rectangle function* defined, for $N \geq 0$, by:

$$\text{rect}_N(n) = u(n) - u(n-N) = \begin{cases} 1 & \text{for } n \in \{0, \cdots, N-1\} \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

- The *sine function* defined by:

$$x(n) = x_0 \sin(2\pi f_0 n + \phi) \tag{2.17}$$

- The *complex exponential* defined by:

$$x(n) = x_0 \exp(2j\pi f_0 n) \tag{2.18}$$

- The *truncated sine function* defined by:

$$x(n) = x_0 \sin(2\pi f_0 n + \phi) \times \text{rect}_N(n) \tag{2.19}$$

- The *truncated complex exponential* defined by:

$$x(n) = x_0 \exp(2j\pi f_0 n) \times \text{rect}_N(n) \tag{2.20}$$

A discrete-time signal will be referred to as either the set $\{x(n)\}$ of its values, or by its generic element $x(n)$ or x_n, depending on the context.

Example 2.3 (Basic signals)
Write a program designed to create and plot basic signals.

HINT: the program `basicfct.m` plots a few basic signals, which are shown in Figure 2.10.

```
%===== BASICFCT.M
N=20; mtime=[0:N-1];
impuls=eye(1,N);                 % Unit Pulse
untstep=ones(1,N);               % Unit Step
f0=.1; fsin=sin(2*pi*f0*mtime); % Sinusoid
```

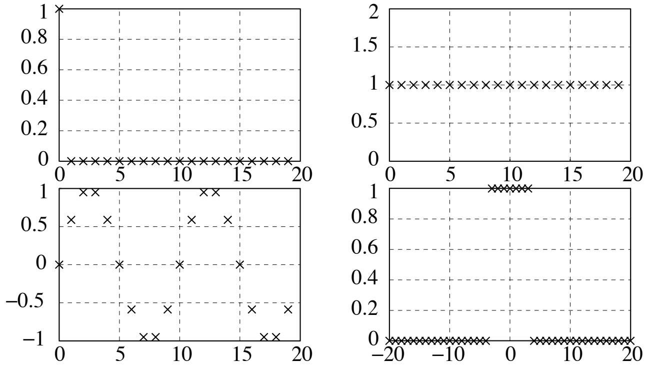


Figure 2.10 – *Basic functions*

```
P=3; tps2=[-N:N];                  % 2P+1 sample rect.
porteP=[zeros(1,N-P) ones(1,2*P+1) zeros(1,N-P)];
subplot(221); plot(mtime,impuls,'x'); grid
subplot(222); plot(mtime,untstep,'x'); grid
subplot(223); plot(mtime,fsin,'x'); grid
subplot(224); plot(tps2,porteP,'x'); grid
```

■

2.3 Spectral representation

The main goal in the spectral study of a signal is to find out how to decompose this signal as a sum of sines. To evaluate the importance of the $\cos(2\pi f_0 n)$ component with the frequency f_0 in the $x(n)$ signal, the first idea would be to calculate:

$$Q(f_0) = \sum_{n \in \mathbb{Z}} x(n) \cos(2\pi f_0 n)$$

which can be interpreted as a quantity that measures how similar the sequences $\{x(n)\}$ and $\{\cos(2\pi f_0 n)\}$ are. It is exactly what the *Discrete-Time Fourier Transform* (definition 2.2) does, as well, in fact, as the Fourier transform for continuous-time functions.

2.3.1 Discrete-time Fourier transform (DTFT)

The sampling period T_s appears in the DTFT's expression in definition 2.4.

Definition 2.4 (DTFT) *The discrete-time Fourier transform of a sequence* $\{x(n)\}$ *is the function of the real variable* f*, periodic with period* 1*, defined by:*

$$X(f) = \sum_{n=-\infty}^{+\infty} x(n)\exp(-2j\pi nf) \tag{2.21}$$

As you can see, we need only impose $FT_s = f$ and replace $x(nT_s)$ by $x(n)$ to go from 2.4 to 2.21[1].

Definition 2.4 calls for a few comments: it can be proven (see ref. [27]) that if $\{x(n)\}$ is summable ($\sum_n |x(n)| < +\infty$), the series (2.21) converges uniformly to a continuous function $X(f)$. However, if $\{x(n)\}$ is square summable ($\sum_n |x(n)|^2 < +\infty$) without having a summable modulus, then the series converges in quadratic mean. There can be no uniform convergence.

Because of its periodicity, the DTFT is plotted on an interval of length 1, most often the intervals $(-1/2, +1/2)$ or $(0, 1)$.

Example 2.4 (DTFT of the rectangle function)
Let $\mathrm{rect}_N(n)$ be the signal given by 2.16. Its DTFT is:

$$\begin{aligned} X(f) &= \sum_{n=0}^{N-1} e^{-2j\pi nf} = 1 + \cdots + e^{-2j\pi(N-1)f} \qquad (2.22)\\ &= \begin{cases} N & \text{for } f = 0 \bmod 1 \\ \dfrac{1-e^{-2j\pi Nf}}{1-e^{-2j\pi f}} = e^{-j\pi(N-1)f}\dfrac{\sin(N\pi f)}{\sin(\pi f)} & \text{for } f \neq 0 \bmod 1 \end{cases} \end{aligned}$$

The $e^{-j\pi(N-1)f}$ is of modulus 1, and only has influence on the phase of $X(f)$ (Figure 2.11).

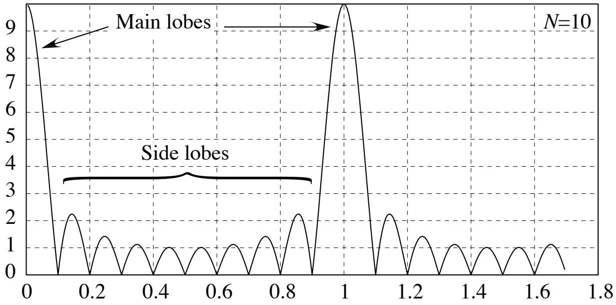


Figure 2.11 – *Modulus of the DTFT of the rectangle signal for* $N = 10$

[1] $X(F)$, which refers to the FT in 2.4 must not be confused with $X(f)$, the DTFT.

$|\sin(N\pi f)/\sin(\pi f)|$ shows one main lobe, with a width of $2/N$ and side lobes with a width $1/N$. We will often deal with this signal again, particularly when observing a signal assumed to be of infinite duration, over a finite number N of values, since it amounts to multiplying it by a rectangle with a duration of N.

Starting off from $X(f)$, how can we go back to $x(n)$? One possible answer is given in the following result.

Theorem 2.2 (Inverse DTFT) *If $X(f)$ is a periodic function with period 1, and if $\int_0^1 |X(f)|^2 df < +\infty$, then $X(f) = \sum_n x(n)e^{-2j\pi nf}$, where the $x(n)$ coefficients are given by:*

$$x(n) = \int_{-1/2}^{1/2} X(f)e^{2j\pi nf}\,df \tag{2.23}$$

Relation between the FT and the DTFT

First let us once again consider the reconstruction formula of a real signal $x(t)$ from its samples $x_s(n)$:

$$x(t) = \sum_n x_s(n)h_B(t - nT_s) \quad \text{with} \quad h_B(t) = \frac{\sin(2\pi Bt)}{\pi F_s t} \tag{2.24}$$

F_s refers to the sampling frequency and B to the bandwidth of the signal $x(t)$. We will assume $F_s \geq 2B$. The frequency, expressed in Hz, will be denoted by F, the normalized frequency (no dimension) by f and the sampling period by $T_s = 1/F_s$.

In practice it is often needed to find the Fourier transform using the DTFT of $x_s(n)$, the frequency F_s and the band B. We get:

$$X(F) = \mathbb{1}(F \in (-B, B)) \sum_{k=-\infty}^{+\infty} X(F - kF_s)$$

The Poisson formula 2.4 leads us to:

$$\begin{aligned} X(F) &= T_s\mathbb{1}(F \in (-B, B)) \sum_{n=-\infty}^{+\infty} x_s(n)e^{2j\pi nF/F_s} \\ &= T_s\mathbb{1}(F \in (-B, B))X_s(F/F_s) \end{aligned}$$

where $X_s(f)$ refers to the DTFT of $x_s(n)$. What should be remembered is that the FT of $x(t)$ is obtained:

– by calculating the DTFT of $x_s(n)$;

- by dividing the amplitude by F_s;
- by multiplying the frequency axis by F_s;
- and by limiting the frequency band to the interval $(-B, B)$.

Conversely, the DTFT of $x_s(n)$ is obtained:

- by calculating the FT of $x(t)$;
- by multiplying the amplitude by F_s;
- by dividing the frequency axis by F_s;
- and by periodizing with period 1.

$$X_s(f) = F_s \sum_{k=-\infty}^{+\infty} X\left((f-k)F_s\right) \tag{2.25}$$

The value of B is often omitted, and implicitly $B = F_s/2$. For example, the MATLAB® function `soundsc(x,Fs)`, produces the signal in the band $(-F_s/2, F_s/2)$ using the sequence `x` and the value `Fs` for the sampling frequency.

The discrete-time Fourier transform's main properties are summarized in Appendix A2.

As in the continuous-time case, we have the Parseval's formula:

$$\sum_{n=-\infty}^{+\infty} |x(n)|^2 = \int_{-1/2}^{1/2} |X(f)|^2 df \tag{2.26}$$

and the conservation of the dot product:

$$\sum_{n=-\infty}^{+\infty} x(n)y^*(n) = \int_{-1/2}^{1/2} X(f)Y^*(f) df \tag{2.27}$$

Because the left member of 2.26 is, by definition, the signal's energy, $|X(f)|^2$ represents the energy's distribution along the frequency axis. It is therefore called the *energy spectral density* (*esd*), or *spectrum*. In the literature, this last word is associated with the function $|X(f)|$. If $X(f)$ is included, this adds up to three definitions for the same word. But in practice, this is not important, as the context is often enough to clear up any ambiguity. It should be pointed out that the two expressions $|X(f)|$ and $|X(f)|^2$ become proportional if the decibel scale is used, by imposing:

$$S_{dB}(f) = 20 \log_{10} |X(f)| \tag{2.28}$$

Example 2.5 (Inverse DTFT of a rectangle)
Let $X(f) = \mathbb{1}(f \in (-b, b))$ be a periodic function with period 1 and $0 < b < 1/2$:

1. Determine the sequence $\{x(n)\}$ that has $X(f)$ as its DTFT.
2. Using this result, find the sequence $y(n)$ that has $Y(f) = (X(f - f_0) + X(f + f_0))/2$ as its DTFT.

HINT:

1. By using relation 2.23, we get:

$$x(n) = \int_{-b}^{b} e^{2j\pi nf}\, df = \frac{1}{2j\pi n}\left[e^{2j\pi nf}\right]_{-b}^{b} = \frac{\sin(2\pi nb)}{\pi n}$$

   $\{x(n)\}$ is a *non-causal* sequence consisting of an infinity of terms.

2. Because of the linearity and modulation properties:

$$y(n) = x(n)\frac{e^{2j\pi nf_0} + e^{-2j\pi nf_0}}{2} = x(n)\cos(2\pi f_0 n)$$

The sequence $y(n)$ also has an infinity of non-zero values. ■

Exercise 2.2 (Time domain hermitian symmetry)
Consider a signal $x(n)$ such that $x(n) = x^*(-n)$. Notice that $x(0)$ is real.

1. Show that its DTFT $X(f)$ is real.
2. Determine the expression of the DTFT $Y(f)$ of the sequence defined by:

$$y(n) = \begin{cases} x(n) & \text{for} \quad n > 0 \\ x(0)/2 & \text{for} \quad n = 0 \\ 0 & \text{otherwise} \end{cases}$$

   Using $Y^*(f)$, find the relation between $X(f)$ and $Y(f)$.

2.3.2 Discrete Fourier transform (DFT)

Definition of the discrete Fourier transform

A computer calculation of the DTFT, based on the values of the samples $x(n)$, imposes an infinite workload, because the sequence is made up of an infinity of terms, and because the frequency f varies continuously on the interval $(0, 1)$. This is why, digitally speaking, the DTFT does not stand a chance against the *discrete Fourier transform*, or *DFT*. The DFT calculation is limited to a finite number of values of n, and a finite number of values of f.

The digital use of the DFT has acquired an enormous and undisputed practical importance with the discovery of a fast calculation method known as the *fast Fourier transform*, or *FFT*. The algorithm for the FFT can be found in paragraph 2.4.

Consider the finite sequence $\{x(0), \ldots, x(P-1)\}$. Using definition 2.21, its DTFT is expressed as follows:

$$X(f) = \sum_{n=0}^{P-1} x(n) e^{-2j\pi n f}$$

where $f \in (0, 1)$. In order to obtain the values of $X(f)$ using a calculator, only a finite number N of values for f are taken. The first idea that comes to mind is to take N values, uniformly spaced-out between 0 and 1, meaning that $f = k/N$ with $k \in \{0, \ldots, N-1\}$. This gives us the N values:

$$X(k/N) = \sum_{n=0}^{P-1} x(n) e^{-2j\pi nk/N} \tag{2.29}$$

In this expression, P and N play two very different roles: N is the number of points used to calculate the DTFT, and P is the number of observed points of the temporal sequence. As we will see later on, N influences the *precision* of the plotting of $X(f)$, whereas P is related to what is called the *frequency resolution*.

In practice, P and N are chosen so that $N \geq P$. We then impose:

$$\tilde{x}(n) = \begin{cases} x(n) & \text{for } n \in \{0, \ldots, P-1\} \\ 0 & \text{for } n \in \{P, \ldots, N-1\} \end{cases}$$

Obviously:

$$X(k/N) = \sum_{n=0}^{P-1} x(n) e^{-2j\pi nk/N} = \sum_{n=0}^{N-1} \tilde{x}(n) e^{-2j\pi nk/N}$$

Because the sequence $x(n)$ is completed with $(N-P)$ zeros, an operation called *zero-padding*, in the end we have as many points for the sequence $\tilde{x}(n)$ as we do for $X(k/N)$. Choosing to take as many points for both the temporal sequence and the frequential sequence does not restrict in any way the concepts we are trying to explain. This leads to the definition of the *discrete Fourier transform*.

Definition 2.5 *Let $\{x(n)\}$ a N-length sequence. Its discrete Fourier transform or DFT is defined by:*

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \qquad k \in (0, 1, \ldots N-1) \tag{2.30}$$

$$\textit{where } W_N = e^{-2j\pi/N} \tag{2.31}$$

is an N-th root of unity, that is to say such that $W_N^N = 1$. The inverse formula, leading from the sequence $\{X(k)\}$ to the sequence $\{x(n)\}$, is:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk} \tag{2.32}$$

To show 2.32, you need to calculate its second member by replacing $X(k)$ by 2.30 and using the following equality:

$$g(n) = \frac{1}{N} \sum_{k=0}^{N-1} e^{2j\pi kn/N} = \begin{cases} 1 & \text{for } n = 0 \bmod N \\ 0 & \text{otherwise} \end{cases} \tag{2.33}$$

With MATLAB®, the `fft` function uses the fast calculation algorithm for the DFT. This is the proper syntax:

```
xf=fft(xt,N)
```

The resulting N-length sequence `xf` is the DFT of the P-length $(N \geq P)$ sequence `xt` (2.29).

If parameter N is missing, it is chosen equal to P. Although the function `fft` allows the calculation of the values of the DFT for any number N of frequency points, N should be taken equal to a power of 2 to reduce the computation time[2].

Exercise 2.3 (Comparing computation speeds)
Write a program that compares the respective speeds of the direct calculation using the expression $\sum x(n) \exp(-2j\pi nf)$ and the FFT calculation. Look into the use of the functions `tic`, `toc`, `etime`... for purposes of measuring computation times.

Use of the DFT to plot and study the properties of the DTFT

As it was said before, the DFT is used to digitally determine the values of the DTFT. The more precise the plotting of the DFT is, the higher the number of frequency points has to be.

[2] The `N=nextpow2(P)` function returns the closest power of 2 greater than P.

Exercise 2.4 (Spectrum of the triangle function)
Consider the triangle function defined by `sig=[1:P P-1:-1:0]`. This function is real. Using the FFT, digitally verify hermitian symmetry properties by plotting:

1. The modulus and the phase of its DTFT for $P = 10$.

2. The imaginary and real parts of the DTFT.

Example 2.6 (Time delay properties)
Let $\{x(n)\}$ be a zero signal outside the $\{-n_0, \ldots, n_1\}$ interval where n_0 and n_1 are two positive integers, and let $y(n)$ be defined by $y(n) = x(n - n_0)$, obtained by a time-shift of n_0 samples:

1. Determine the DTFT of $\{x(n)\}$, expressing it as a function of the DTFT of $\{y(n)\}$.

2. Write a program that checks the previous result for $n_0 = 5$. In order to do this, set $\{x(n)\}$ equal to 1 between -5 and 5, and $y(n) = x(n - 5)$. To digitally evaluate the DTFT over 256 frequency points regularly spaced-out in the $(0, 1)$ interval, the `fft` function is used.

HINT:

1. We have:

$$\begin{aligned} X(f) &= \sum_{n=-n_0}^{n_1} x(n) e^{-2j\pi n f} = e^{2j\pi n_0 f} \sum_{k=0}^{n_1+n_0} x(k-n_0) e^{-2j\pi k f} \\ &= e^{2j\pi n_0 f} \sum_{k=0}^{n_1+n_0} y(k) e^{-2j\pi k f} = e^{2j\pi n_0 f} Y(f) \end{aligned}$$

2. $X(f) = \sin(5\pi f)/\sin(\pi f)$. This means that to get the DTFT of $\{x(n)\}$, all you have to do is calculate the DTFT of $\{y(n)\}$ and multiply it by $e^{10j\pi f}$.

The following program can be used to verify this:

```
%===== SHIFTF.M
Lfft=256;           % Length equal to a power of two
f=(0:Lfft-1)/Lfft; % Normalized Freq.
n0=5; n1=5; yt=ones(n1+n0+1,1);
Yf=fft(yt,Lfft);    % DFT of y(n)
Xf=Yf .* exp(2*j*pi*5*f');
subplot(211); plot(real(Xf)); grid
%==== Imag. part roughly zero (temporal symmetry)
subplot(212); plot(imag(Xf)); grid
```

■

Properties of the DFT

The properties of the DFT show strong similarities with those of the DTFT. However, there is an essential difference. In the formulas associated with the DFT, *all the index calculations are done modulo N*. The discrete Fourier transform's main properties are summarized in Appendix A3.

Exercise 2.5 (Circular convolution of the rectangular signal)
Consider the rectangular signal $x(n) = \mathbb{1}(n \in \{0, \cdots, 7\})$. Compare and explain the effects of the following commands (`ifft` is the function used to obtain the inverse DFT):

```
x=ones(1,8); xs=fft(x); xs=xs .* xs; ifft(xs)
```

and:

```
x=ones(1,8); xs=fft(x,16); xs=xs .* xs; ifft(xs)
```

Exercise 2.6 (Delay)
Because of the time shift property, in order to get the L points DFT of a signal that has non-zero values between $-n_0$ and n_1, the sequence's DFT must be calculated on N points and then the delay has to be taken into account, by multiplying the result, term-by-term, by the complex exponential $\exp(2j\pi n_0 k/L)$, where $k \in \{0, \ldots, L-1\}$. This exercise introduces a different method to achieve the same result.

Let $x(n)$ be a signal equal to zero for n outside the set of indices $\{-n_0, \ldots, n_1\}$, where n_0 and n_1 are positive, and let $y(n)$ be the signal defined by:

$$y(n) = \begin{cases} x(n) & \text{for } n \in \{0, \ldots, n_1\} \\ 0 & \text{for } n \in \{n_1+1, \ldots, L-n_0-1\} \\ x(n-L) & \text{for } n \in \{L-n_0, \ldots, L-1\} \end{cases}$$

with $L > n_0 + n_1$. One way of seeing it is to imagine the values of $x(n)$ with negative indices being translated to the right by L points.

1. Calculate the DTFT of $y(n)$ on L points. Conclude.

2. Let $x(n)$ be a signal equal to 1 between -5 and 5, apply the previous result to a program designed to calculate the DTFT of $x(n)$ on 256 points.

The point of exercise 2.6 is to explain that, in order to determine the DFT of a sequence $x(n)$ with a length of N, for L points, with $L > N$, you need to calculate the DFT of the sequence:

$$y(n \bmod L) = x(n)$$

meaning the sequence whose indices are calculated modulo L.

Example 2.7 (Calculating the IDFT using the DFT)
Let $X(k)$ be the DFT of $x(n)$, and let $y(k) = jX^*(k)$ be the sequence resulting from the permutation of the imaginary parts and the real parts of $X(k)$. In other words, $y(k) = X^I(k) + jX^R(k)$, where $X^R(k)$ and $X^I(k)$ refer to the real and imaginary parts of $X(k)$ respectively.

Calculate the DFT of $y(k)$. Use the result to determine a method for calculating the inverse DFT of a sequence using a *direct* DFT function with the real and imaginary parts as its input, and the real and imaginary parts of the IDFT as its output.

HINT: applying the definition of the DFT to the sequence $y(k)$, we get:

$$\begin{aligned} Y(n) &= \sum_{k=0}^{N-1} y(k)e^{-2j\pi nk/N} = j\sum_{k=0}^{N-1} X^*(k)e^{-2j\pi nk/N} \\ &= j\left(\sum_{k=0}^{N-1} X(k)e^{2j\pi nk/N}\right)^* = j\,(Nx(n))^* = N((x^I(n)) + jx^R(n)) \end{aligned}$$

This means that the use of the DFT function on the sequence $jX^*(k)$ leads to the reconstruction of the original sequence $x(n)$ (multiplied by the factor N) with its real and imaginary parts switched.

Let us now assume that we have at our disposal a direct DFT that has two arrays as its input, one for the real part, and the other for the imaginary part of the signal we wish to transform, and that has two arrays as its output, one for the real part, and the other for the imaginary part of the transform, according to the following synopsis:

```
(XR,XI)= dft(xR,xI)
```

To go from this function to the inverse DFT, all we have to do is set the transform as the input, by switching the roles of the real and imaginary parts. The resulting output is the inverse DFT except for a factor $1/N$. This can be expressed as follows:

```
(xI,xR)= dft(XI,XR)
```

In MATLAB®, the `fft` function, used to directly calculate the DFT, has an array of complex numbers as its argument, which means that it is not possible to apply the previous result. MATLAB®'s `ifft` function, in order to calculate the inverse DFT from the direct DFT, uses the conjugation property:

$$x(n) = \frac{1}{N}\left(\sum_{k=0}^{N-1} X^*(k)e^{-2j\pi kn/N}\right)^*$$

The inverse DFT is the conjugate of the conjugate's direct DFT. This can be written `x=conj(fft(conj(X)))` where `fft` is the function calculating the DFT (see next paragraph). ■

2.4 Fast Fourier transform

The *fast Fourier transform*, or *FFT*, first published in 1965 by J. W. Cooley and J. W. Tuckey [24], is a fast DFT calculation technique. The basic algorithm, many versions of which can be found, calculates a number of points N, equal to a power of 2, and the time saved compared with a direct calculation is roughly:

$$\text{gain} = \frac{N}{\log_2(N)}$$

To get a better idea, if $N = 1{,}024$, the FFT is about *100 times faster* than the direct calculation based on the definition of the DFT.

To understand its mechanisms, consider the case $N = 8$. Using the notation $W_N = \exp(-2j\pi/N)$, the DFT can be expressed as the sum of a term related to even rank indices and of a term related to odd rank indices:

$$\begin{aligned} X_k &= \left(x(0) + x(2)W_8^{2k} + x(4)W_8^{4k} + x(8)W_8^{6k}\right) \\ &\quad +W_8^k\left(x(1) + x(3)W_8^{2k} + x(5)W_8^{4k} + x(7)W_8^{6k}\right) \\ &= \left(x(0) + x(2)W_4^{k} + x(4)W_4^{2k} + x(8)W_4^{3k}\right) \\ &\quad +W_8^k\left(x(1) + x(3)W_4^{k} + x(5)W_4^{2k} + x(7)W_4^{3k}\right) \end{aligned} \tag{2.34}$$

A length 8 DFT is thus replaced by two length 4 DFTs. By iterating the process, the DFT's length is divided by two at every step. It takes 10 steps to go from a length 1024 DFT to length 2 DFTs. In our case, the next step is (see Figure 2.12):

$$\begin{aligned} &\left(x(0) + x(4)W_4^{2k}\right) + W_4^k\left(x(2) + x(6)W_4^{2k}\right) = \\ &\qquad\qquad \left(x(0) + x(4)W_2^{2k}\right) + W_4^k\left(x(2) + x(6)W_2^{k}\right) \end{aligned}$$

and:

$$\begin{aligned} &\left(x(1) + x(5)W_4^{2k}\right) + W_4^k\left(x(3) + x(7)W_4^{2k}\right) = \\ &\qquad\qquad \left(x(1) + x(5)W_2^{k}\right) + W_4^k\left(x(3) + x(7)W_2^{k}\right) \end{aligned}$$

Each term is associated with a sum, a subtraction, and a multiplication by a power of W_N. An example for this kind of calculation is detailed in Figure 2.12.

By representing all of the terms in a diagram, the calculation algorithm, Figure 2.13, shows an elementary structure called *butterfly*.

Evaluating the number of operations

As it can be seen in expression 2.34, a length 8 FFT was replaced, in the first step, by two length 4 FFTs. We have to include 8 complex multiplication-addition operations (call *MAC* operations[3]).

[3] The acronym *MAC* is in reference to the Multiplication-ACcumulation operation that can be found in the $s = s + a_i b_i$ algorithm, used to calculate a sum of products $\sum_i a_i b_i$.

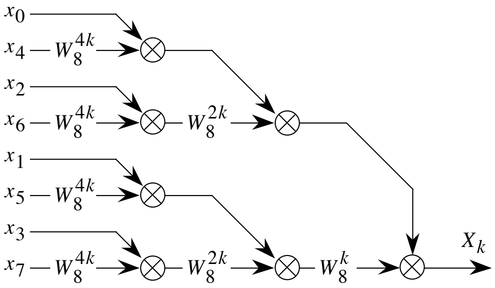


Figure 2.12 – *Calculation of one of the FFT's terms*

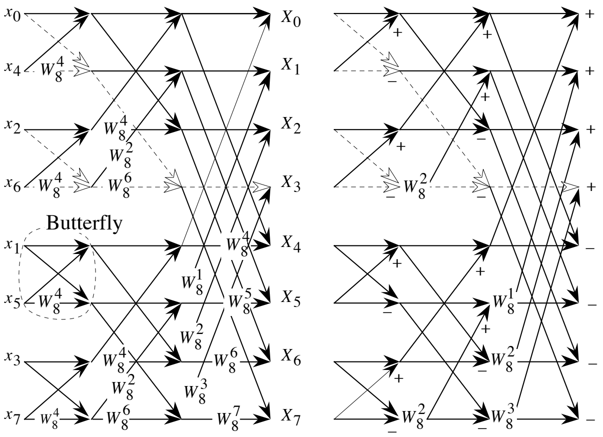


Figure 2.13 – *Calculations of the FFT terms. The dotted arrows show the calculation of X_3*

This result can easily be generalized for a length N DFT, where N equals a power of 2: if C_N is the number of *MAC* operations for the Nth step, $C_N = 2C_{N/2} + N$ leads us to the complexity:

$$C_N = N \times \log_2(N)$$

We also have to include an index calculation phase needed to access the data. Figure 2.13 shows that the indices of the terms x_n appear in an order corresponding to the inverted binary code of n, as it is indicated in the following table. This is called the *bit reverse* access.

Rank	Binary Coding	Reversal	Element
0	000	000	0
1	001	100	1
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

COMMENTS:

- All of the W_N^k terms are displayed in the left part of Figure 2.13. The part on the right was simplified by considering certain values, particularly $W_8^4 = -1$. As you can see, first level processing is limited to adding and subtracting. The second level could also be dealt with in this particular way.

  In most of the FFT calculation programs, using these simplifications allows you to save a little time.

- Processors designed for signal processing have a particular addressing mode, exempting them from actually calculating the indices. The addressing mode is called *bit reverse addressing*.

Exercise 2.7 (FFTs of real sequences)
Consider the real sequence $x(n)$, with $n \in \{0 \ldots N-1\}$. Let $X(k)$ be its DFT. The complex sequence $y(n)$ will be defined by $y(n) = x(2n) + jx(2n+1)$. Let A_k and B_k be the DFTs of the sequences $x(2n)$ and $x(2n+1)$ respectively. By linearity, $y(n)$ has the sequence $Y(k) = A(k) + jB(k)$ as its DFT (notice that $A(k)$ and $B(k)$ may be complex. Therefore $A(k)$ and $B(k)$ are not the real and imaginary parts of $Y(k)$ respectively).

1. By noticing that $x(2n)$ is the real part of $y(n)$ and is therefore equal to $(y(n) + y^*(n))/2$, express $A(k)$ using the term $Y(k)$. Do the same for $B(k)$.

2. Find a method similar to the decomposition given by expression 2.34 to show that $X(k)$ can be expressed as a function of $A(k)$ and $B(k)$. Using this result, write an algorithm that calculates the DFT of a real length N sequence based on a complex length $N/2$ FFT algorithm.

3. Compare the complexities of the previous algorithm and the complex length N FFT algorithm.

Exercise 2.8 (Using the FFT)

What is the purpose of the following program:

```
plot(fft([0 1],128))
set(gca,'AspectRatio',[1 1])
```

Chapter 3

Spectral Observation

The purpose of this chapter is to introduce the reader to the two following fundamental concepts:

- the *accuracy* of the frequency measurement when the DFT is used to evaluate a signal's DTFT. As we will see, this accuracy depends on the number of points used to calculate the DFT;
- the *spectral resolution*, which is the ability to discern two distinct frequencies contained in the same signal. It depends on the observation time and on the weighting windows applied to the signal.

3.1 Spectral accuracy and resolution

3.1.1 Observation of a complex exponential

To illustrate the DFT's use in signal spectrum observation, we will begin with a simple example.

Example 3.1 (Sampling a complex exponential) Consider the sequence resulting from the sampling of a complex exponential $e^{2j\pi F_0 t}$ at a frequency of $F_s = 1/T_s$. If we set $f_0 = F_0/F_s$ and assume it to be $< 1/2$, we get $x(n) = e^{2j\pi f_0 n}$.

1. Determine the DTFT's expression for the sequence $\{x(n) = \exp(2j\pi f_0 n)\}$ where $f_0 = 7/32$ and $n \in \{0, \cdots, 31\}$.
2. Using this result, find the DTFT's values at the points of frequency $f = k/32$, for $k \in \{0, \cdots, 31\}$.
3. Using the `fft` command, display the modulus of the DFT of $\{x(n)\}$.
4. Now let $f_0 = 0.2$. Display the modulus of the DFT of $\{x(n)\}$. How do you explain the result?

HINT:

1. Starting off with definition 2.21 of the DTFT, we get:

$$X(f) = \sum_{n=0}^{N-1} e^{2j\pi f_0 n} e^{-2j\pi f n} = \frac{\sin(N\pi(f-f_0))}{\sin(\pi(f-f_0))} e^{-j\pi(N-1)(f_0-f)} \quad (3.1)$$

   Because a finite duration sequence is all we have at our disposal, the signal's DTFT shows ripples (ratio of the sines). $|X(f)|$ is plotted in Figure 3.1, illustrating this phenomenon. This was achieved with the following program:

```
%===== RESOL1.M
N=32;                % Number of points of the signal
f0=7/32;             % Sine Frequency
npts=512;            % Number of points of the frequency
freqmin=-0.5; freqmax=0.5;
pas=(freqmax-freqmin)/npts;
f=[freqmin:pas:freqmax-pas]; freqM=f-f0;
%===== Direct calculion of the DTFT
fctM=sin(N*freqM*pi) ./ sin(freqM*pi);
plot(f,abs(fctM)); grid
hold on; plot([f0 f0],[0 35]); hold off
```

   in which expression 3.1 is directly used.

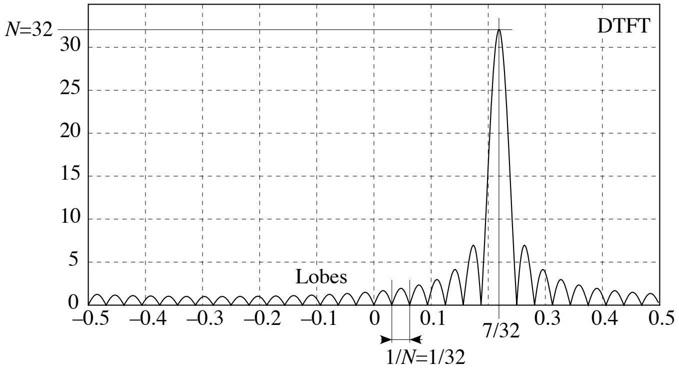


Figure 3.1 – *Modulus of the DTFT of the complex exponential* $f_0 = 7/32$ *with* $N = 32$

2. Because the DFT corresponds to a sampling of the DTFT at frequency points k/N, its values are usually different from zero, except if f_0 is an exact multiple of $1/N$, which is the case for $f_0 = 7/32$. The values of f are given by 0, 1/32, ..., 31/32. We then get $X(k) = 32$ if $k = 7/32$ and 0 otherwise. Type:

```
%===== RESOL2.M
N=32; L=32; freq=(0:L-1)/L;
f0=7/32; xt=exp(2*j*pi*f0*(0:N-1));
xf=fft(xt,L);              % Calculation with the DFT
plot(freq,abs(xf),'x');
%===== The DTFT calculated by FFT is superposed
L=512; freq=(0:L-1)/L; xf=fft(xt,L);
hold on ; plot(freq,abs(xf),':'); grid; hold off
```

This leads to the graph in Figure 3.2.

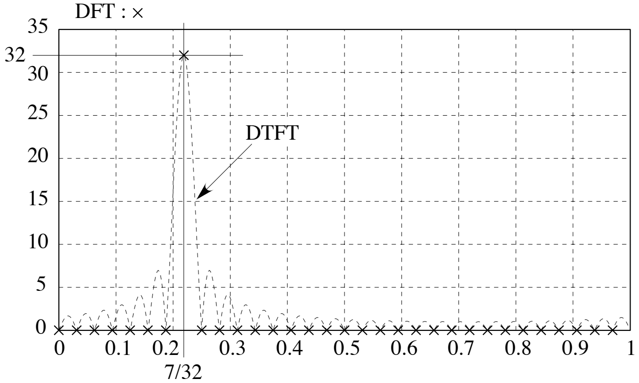


Figure 3.2 – *DFT of the complex exponential when f_0 is a multiple of* $1/N$

A "peak" is observed (this is actually the only non-zero value of $X(k)$), with an amplitude of 32 at a frequency of $7/32$, and all the other frequency points have a zero spectrum. This result, which seems to agree with what would be expected of a infinite duration complex exponential with only one peak, is rather exceptional.

3. Figure 3.3 results from imposing $f_0 = 0.2$, and as it clearly shows, it is quite different from a single "peak". An explanation of this can be found in paragraph 3.1.4 which deals with the subject of windowing.

■

3.1.2 Plotting accuracy of the DTFT

As we have just seen, the DFT is all we have at our disposal to plot the DTFT, or rather its modulus. The previous example is a good illustration of a number of important properties, the first of which is:

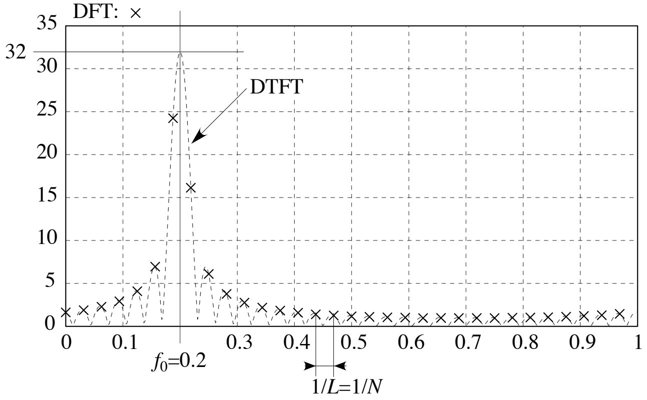


Figure 3.3 – *DFT of the complex exponential when f_0 is not a multiple of* $1/N$

For those frequencies that are not a multiple of $1/L$*, where L is the number of calculated DFT points, a pure sine appears in the form of several non-zero values. The value with the highest modulus is close to the actual frequency.*

It should be noted that the gap between the frequency f_0 and the frequency associated to the maximum of the $L = 32$ values of the DFT's modulus is, in the worst case, equal to $1/L$. This leads us to the following rule:

If L refers to the number of DFT calculation points, the frequency accuracy is equal to $1/L$*. For signals sampled at a frequency of F_s (in Hz), this leads to a accuracy of F_s/L Hz.*

3.1.3 Frequency resolution

Accuracy must not be confused with the ability to distinguish (or *separate*) two close frequencies in a signal. One possible definition of the *frequency resolution* is the minimum difference between the two sine frequencies with different amplitudes, necessary to "observe" an attenuation greater than 3 dB between their two maximums.

As we saw on page 68, limiting ourselves to handling a number of values no greater than N causes *lobes* to appear in the sine spectrum. The main

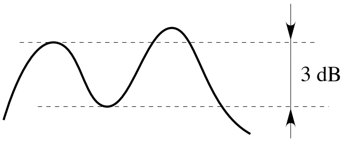


Figure 3.4 – *Separation of frequencies*

lobe's width is equal to $2/N$. This means that if $x(n)$ contains two sines the frequencies of which are separated by less than $1/N$, their two main lobes will be so close that it will be difficult to distinguish them by observing the spectrum. This is even more true when their amplitudes are very far apart.

Resolution and noise

As we will see in Chapter 10, there is no point in talking about frequency resolution in the absence of noise. Consider observations made without noise and assume we have 100 measurement points of the signal $x(n) = A_1 \cos(2\pi f_1 n + \phi_1) + A_2 \cos(2\pi f_2 n + \phi_2)$. To determine, from the values of $x(n)$, the two frequencies, the two amplitudes and the two phases, we have to solve the following system of six equations with six unknowns:

$$\left\{\begin{array}{lcl} A_1 + A_2 & = & x(0) \\ A_1 \cos(2\pi f_1 + \phi_1) + A_2 \cos(2\pi f_2 + \phi_2) & = & x(1) \\ & \vdots & \\ A_1 \cos(10\pi f_1 + \phi_1) + A_2 \cos(10\pi f_2 + \phi_2) & = & x(5) \end{array}\right.$$

The 94 remaining values must be consistent with the result! It should be noted that the precision of the result is limited only by the calculator's precision, and that no conditions have to be met regarding the difference between f_1 and f_2. And there's no point in using the DTFT calculation!

However, if there is some noise, the observed values of $x(n)$ are "riddled with errors". The statistical estimation theory tells us that it is better to use all of the values, calculating some sort of a mean value. This is precisely what the DTFT does. Separating f_1 and f_2 now depends on the difference between f_1 and f_2, but also on the desired signal-to-noise ratio.

If $F_s = 1/T_s$ refers to the sampling frequency, we have:

The frequency resolution R is expressed in Hz. Its has the same order of magnitude as F_s/N, which is also the inverse of the total observation time $T = NT_s$.

Without additional informational, frequency differences of less than $F_s/N = 1/T$ should not be interpreted when studying a spectrum! In the literature, the quantity $R = F_s/N = 1/T$ is called the *Fourier limit.*

As an example, type the following program:

```
%===== RESOLFREQ.M
N=32; L=128; freq=(0:L-1)/L;
%===== First frequency
f0=.2; xt0=exp(2*j*pi*f0*(0:N-1)); xf0=fft(xt0,L);
%===== Second frequence = 0.23
f1=.23; xt1=exp(2*j*pi*f1*(0:N-1)); xf1=fft(xt1,L);
subplot(311); plot(freq,abs([xf0' xf1' (xf0+xf1)']));
grid
%===== Second frequency = 0.22
f1=.22; xt1=exp(2*j*pi*f1*(0:N-1)); xf1=fft(xt1,L);
subplot(312); plot(freq,abs([xf0' xf1' (xf0+xf1)']));
grid
%===== Third frequency = 0.21
f1=.21; xt1=exp(2*j*pi*f1*(0:N-1)); xf1=fft(xt1,L);
subplot(313); plot(freq,abs([xf0' xf1' (xf0+xf1)']));
grid
```

Figure 3.5 shows the modulus of the DTFT for the sum of two sines, for three frequency pairs. In the first case, $f_0 = 0.2$ and $f_1 = 0.23$, the presence of two sines can be shown, whereas it is impossible in the other cases.

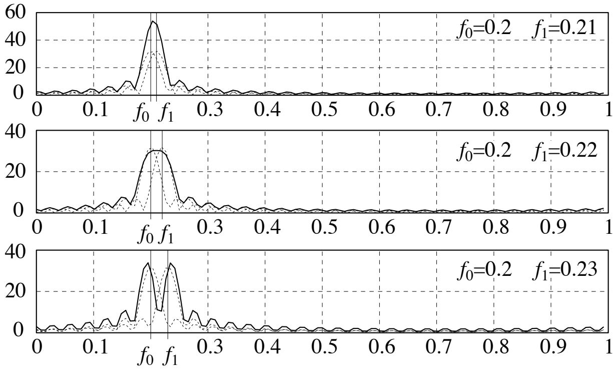


Figure 3.5 – *Frequency resolution: the closer the two frequencies are, the harder it is to distinguish their peaks*

The $R \times T$ product plays the role of a *merit factor* when using the DTFT to search for frequencies. For a given resolution R, choosing T so as to have $R \times T \geq 3$ usually allows an easy separation of the frequencies.

Exercise 3.1 (Studying the resolution)
Consider the signal $x(n)$, sum of two real sines with a frequency of f_0 and $f_1 = f_0 + \Delta f$ and amplitudes of $a_0 > 0$ and $a_1 > 0$ respectively.

1. Using $g_N(f) = \sin(N\pi f)/\sin(\pi f)$, give an expression of $X(f)$.

2. Let $Nf_0 \gg 1$, $Nf_1 \gg 1$ and $N|\Delta f| \gg 1$. Use these inequalities to show that $|X(f)|$ has two maximums close to the 2 frequencies f_0 and f_1.

3. Write a program that displays the signal's spectrum for a given $a = a_1/a_0$ dB ratio, and a given phase shift. Change the difference Δf from $1/N$ to $2/N$ for $N = 32$ and $f_0 = 0.2$. Without changing any other parameters, compare the two resolutions corresponding to $\Phi = 0$ and $\Phi = \pi/2$.

3.1.4 Effects of windowing on the resolution

Rectangular windows

Limiting the number of samples N of a signal can be interpreted as the term-by-term multiplication of the signal by the sequence $w_N(n) = \mathbb{1}(n \in \{0, \ldots, N-1\})$. This sequence is called a *rectangular window*. The same signal was called a "rectangle" in the previous chapter.

From a spectral perspective, this multiplication, or *weighting*, is equivalent to convoluting the DTFT of $x(n)$ with the DTFT $W_N(f)$ of the sequence $w_N(n)$. This can be written as follows:

$$\{x(n) \times w(n)\} \rightarrow (X \star W_N)(f)$$

where $W_N(f)$ is expressed (formula 2.22):

$$W_N(f) = \frac{\sin(N\pi f)}{\sin(\pi f)} e^{-j\pi(N-1)f}$$

The effect of this convolution operation is to cause unwanted ripples to appear in the spectrum.

The concept of windows

Generally speaking, a *window* is a sequence of coefficients used to weight a signal. A relatively detailed study of the windows used for signal processing can be found in [43]. Usually, when the frequency resolution is improved:

– the main lobe grows narrower;

– and the side lobes become smaller.

Unfortunately, reducing the height of the side lobes always means widening the main lobe. A compromise must therefore be made between these effects. In the following exercise, which illustrates these properties, we will only be using the *Hamming window*, one of the most commonly used windows. Its expression is:

$$w_h(n) = \begin{cases} 0.54 - 0.46\cos(\frac{2\pi n}{N}) & \text{when } n \in \{0, \cdots, N-1\} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Exercise 3.2 (Effect of the Hamming windowing)
Consider a length $N = 32$ sample of a complex exponential $x(n)$ with a frequency of $f_0 = 0.2$ and an amplitude of $A = 1$. Each sample is multiplied by $c_h w_h(n)$, where $w_h(n)$ refers to the Hamming window and c_h is a constant we have to determine.

1. Calculate, for any window, the constant c_h such that the maximum amplitude of the DFT of the windowed signal at f_0 is equal to A.

2. Write a program that displays the DTFT of $x(n)$ for the rectangular windowing and the Hamming windowing.

3. For both windows, check the width of the main lobe and the height of the side lobe (the lobe's height will be expressed in dB compared to the height of the main lobe).

4. We want to distinguish, in a signal sampled at 1,000 Hz, two sines of the same amplitude. Use the previous plot to find an order of magnitude for the resolution of the two windows that were studied.

5. We want to distinguish, in a signal sampled at 1,000 Hz, two sines with an amplitude ratio now worth 25 dB. Find an order of magnitude for the windows that were studied.

In practice, the frequency resolution for sines of the same amplitude is roughly equal to $1/N$ when using a rectangular window. When the amplitude ratio is no longer equal to 1, the resolution depends on which analysis window is chosen. Exercise 3.2 shows that the Hamming window leads to a resolution that is not as good as the one obtained with the rectangular window, for an amplitude ratio of 0 dB, but this phenomenon is reversed for an amplitude ratio of 25 dB.

A few windows

The following table gives a few characteristics for the most commonly used windows (see Figure 3.6). Δ is the main lobe's width and A_{dB} is the attenuation, in dB, of the first side lobe, compared to the main lobe's height. The results of this table can be found using a MATLAB® program of the type (see Figure 3.6):

```
%===== ONEWIN.M
%      Blackman
N=10; w=0.42-0.5*cos(2*pi*(0:N-1)/N)+0.08*cos(4*pi*(0:N-1)/N);
w=w/sum(w);          %===== Gain in 0 equal to 1
ws=fft(w,1024);
plot((0:1023)/1024,20*log10(abs(ws)))
set(gca,'xlim',[0 .5],'ylim',[-100 0]); grid
%===== To measure freq. click once on each max.
[xm,ym]=ginput(2)
```

Type	Expression for $n \in \{0, \dots, N-1\}$	Δ	$A_{\text{dB}} \approx$
Rectangular	$\mathbb{1}(n \in \{0, \dots, N-1\})$	$2/N$	-13 dB
Triangular	N-width Triangle	$4/N$	-25 dB
Hann	$0.5 - 0.5\cos(\frac{2\pi n}{N})$	$4/N$	-31 dB
Hamming	$0.54 - 0.46\cos(\frac{2\pi n}{N})$	$4/N$	-41 dB
Blackman	$0.42 - 0.5\cos(\frac{2\pi n}{N}) + 0.08\cos(\frac{4\pi n}{N})$	$6/N$	-61.5 dB

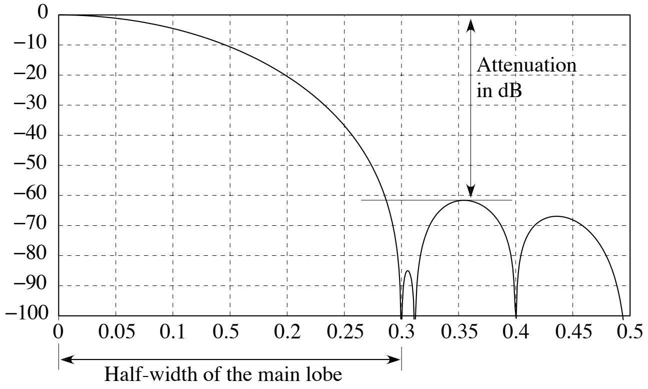


Figure 3.6 – *The Blackman window parameters*

Periodic and symmetrical window

Consider, for $n \in \{0, \cdots, N-1\}$, the two following expressions of the Hamming window:

$$w_P(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right) \text{ and } w_S(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right)$$

The first one, indexed with a P, is periodic with period N, that is to say $w_P(0) = w_P(N)$. It is used, among other things, as a weighting window for the spectral analysis, of length N portions of a signal. The second one, indexed with an S, is symmetrical in the sense that $w_S(0) = w_S(N-1)$, $w_S(1) = w_S(N-2)\ldots$ As we will see, it is particularly used as a weighting window in the case of length N FIR filter design (see paragraph 4.7, page 133).

If you have the MATLAB® *signal toolbox* at your disposal, type `help hamming`. Depending on what version you own, you may or may not have the choice between periodic windows and symmetrical windows.

3.2 Short term Fourier transform

The Fourier transform "compares" the signals to the eternal exponentials by calculating a mean on the time axis. It is therefore better suited for the study of phenomena that vary little in time than it is for brief, transitory phenomena. This does not mean, however, that information is lost, because the Fourier transform is bijective under the conditions expressed in the introduction chapter. Consider, for example, the signal $x(t)$ made up of two consecutive portions of sines with durations of T_1 and T_2 and frequencies of $f_1 = 0.1$ and $f_2 = 0.2$ (Figure 3.7). This signal can be created by the following program:

```
%===== TWOSIN1.M
T1=512; T2=256;                 % Respective durations
tps1=[0:T1-1];tps2=[0:T2-1]; tps=[tps1 T1+tps2];
f1=0.1; x1=sin(2*pi*f1*tps1);
f2=0.2; x2=sin(2*pi*f2*tps2);
x=[x1 x2]; plot(tps,x); grid    % Plotting of the 2 sinusoids
set(gca,'xlim',[384 576])
```

The Fourier transform $X(f)$ of the complete signal "contains" the information regarding the order in which the two sines appear. However this information's interpretation is difficult, because it is found, not very explicitly, in the transform's phase. Therefore, by limiting ourselves to the visualization of the modulus of $X(f)$, there is no way for us to know that f_1 comes before f_2. This can be illustrated by typing the following program:

```
%===== SPECCT1.M
% Plotting of the modulus and phase of the signal x
% defined in program TWOSIN1.M
```

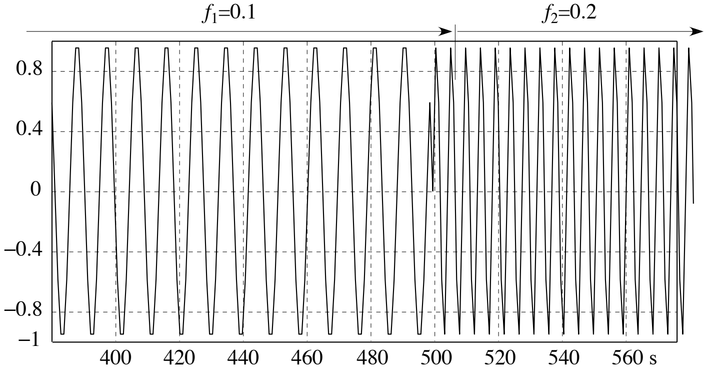


Figure 3.7 – *Two portions of sines*

```
Lfft=1024; freq=(0:Lfft-1)/Lfft;
xf=fft(x,Lfft); xfa=abs(xf); xfph=angle(xf);
subplot(211); plot(freq,xfa); grid
set(gca,'xlim',[0 0.5],'ylim',[0 max(xfa)]);
subplot(212); plot(freq,xfph); grid
set(gca,'xlim',[0 0.5],'ylim',[-pi pi]);
```

Figure 3.8 shows two peaks at frequencies 0.1 and 0.2, but it does not tell us which one comes first.

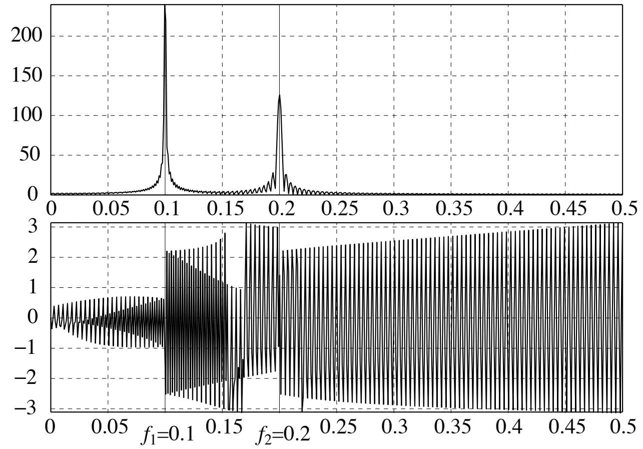


Figure 3.8 – *Modulus and phase for the two portions of sines*

Type:

```
%===== SPECCT2.M
```

```
% x1 and x2 are defined in TWOSIN1.M
xinv=[x2 x1]; xinvf=abs(fft(xinv,Lfft));
plot(freq,xinvf); grid
set(gca,'xlim',[0 0.5],'ylim',[0 max(xinvf)]);
```

The resulting spectrum is almost identical to the previous one. However, if the time interval is “cut up” in N_{si} sub-intervals with a duration of P_i, and if Fourier transforms are performed on each of these sub-intervals, the information concerning the order of the frequencies becomes clear. This leads us to the concept of *short term Fourier transform*, or *STFT*. At the end of the previous program, type:

```
%===== SPECCT3.M
% T1,T2, x defined in TWOSIN1.M
nfft=1024; freq=[0:nfft-1]/nfft;
%=====
nsi=8; npt=fix((T1+T2)/nsi);
xs=zeros(npt,nsi); xs(:)=x(1:npt*nsi);
xsf=abs(fft(xs,nfft)); xsf=xsf(1:nfft/2,:);
mtime=[0:npt:npt*nsi-1];
subplot(211); imagesc(mtime,freq(1:nfft/2),xsf);
set(gca,'xlim',[0 700])
%=====
nsi=32; npt=fix((T1+T2)/nsi);
xs=zeros(npt,nsi); xs(:)=x(1:npt*nsi);
xsf=abs(fft(xs,nfft)); xsf=xsf(1:nfft/2,:);
mtime=[0:npt:npt*nsi-1];
subplot(212); imagesc(mtime,freq(1:nfft/2),xsf);
set(gca,'xlim',[0 700])
```

The spectra, which can be displayed using the **imagesc** command (see Chapter 6), are represented in Figure 3.9 for two values of N_{si}.

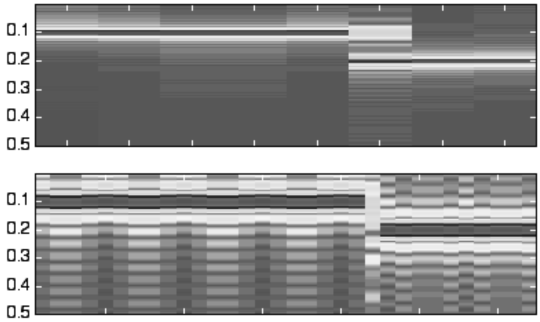


Figure 3.9 – *Spectrum for $N_{si} = 8$ (above) and $N_{si} = 32$ (below)*

surf or **mesh** can also be used for 3D graphs.

It is clear that by following the time axis, the STFT tells us the order of the frequencies used in the signal. By comparing the two figures, we notice that the smaller the number of points K_i in a sub-interval:

- the easier it is to locate the position on the time axis of $T_1 = 512$, corresponding to the frequency change;
- the harder it is to locate the positions 0.1 and 0.2 on the frequency axis because of the width of the main lobes.

Let P_i be the number of points in an interval. The following comments can be made:

- Because the DTFT calculates a “mean” of P_i values, choosing a high value of P_i causes an intense smoothing of the signal’s fluctuations in time. This means that the time transitions cannot be located precisely.
- On the other hand, a high value of P_i gives every DTFT more calculation points. Therefore the width of the lobes (roughly equal to $1/P_i$) decreases and the frequency peaks appear more clearly.

Given the sampling frequency $F_s = 1/T_s$, the frequency resolution is roughly equal to $R_F = F_s/P_i$, while the time resolution is roughly equal to $R_T = P_i T_s$, meaning that the product of the two remains roughly equal to 1.

When using the short term Fourier transform (STFT), improving time resolution decreases frequency resolution.

Exercise 3.3 (Short term Fourier transform)

The incTF1.mat and incTF2.mat we are going to use are supposed to come from the sampling at $F_s = 1{,}000$ Hz of the sum of a certain number of frequential components with different durations. To access these files, execute the two following programs:

```
%===== GENE1.M
T=0.35; Fs=1000; NT=fix(Fs*T); tp=(0:NT-1); xt=zeros(1,NT);
fq=[113 247 327 413]/Fs; org=fix([0 0 0.030 0.150]*Fs);
dur=fix([0.350 0.050 0.200 0.200]*Fs); amp=[1 1.7 1.9 1.8];
for ii=1:4
    xc=amp(ii)*cos(2*pi*fq(ii)*tp);
    ti=org(ii)+1;tf=org(ii)+dur(ii);
    xt(ti:tf)=xt(ti:tf)+xc(ti:tf);
end
save incTF1 xt Fs
```

```
%===== GENE2.M
T=0.35; Fs=1000; NT=fix(Fs*T); tp=(0:NT-1)/Fs;
f0=250; fm=3; beta=6;
phi=f0*tp+beta*sin(2*pi*fm*tp); xt=cos(2*pi*phi);
save incTF2 xt Fs
```

1. Write a short term Fourier transform function. It will be named:

   ```
   [spec,tps]=tfct(xt,Lb,ovlp,Lfft,win)
   ```

   where:

   – `xt` is the signal,
   – `Lb` the length (number of samples) of the blocks,
   – `ovlp` the number of overlapping samples,
   – `Lfft` the length of the FFT,
   – `win` the window type,
   – `spec` the complex spectrogram,
   – `tps` the normalized time.

2. Load one of the two signals and display its chronogram.

3. Write a program designed to perform a time/frequency analysis of the signal. The time interval will be cut up in sub-intervals of the same length, with an overlapping coefficient of 50%. The `contour` function will be used for displaying the results.

Exercise 3.4 (Visualizing the aliasing with the STFT)
Consider the signal 2.11 created in the paragraph on page 60 and illustrating the the aliasing phenomenon by listening to the created signal. Visualize the evolution of this signal's spectrum using the `mesh` function, or in a 2D graph by using the `contour` function. The values $T = 2$ and $\lambda = 2{,}000$ will be taken. In order to achieve this, the signal will be cut up in "slices" with a length of 100 samples, on which FFTs will be performed. The blocks should not overlap. A Hamming window can be used before performing the FFT.

3.3 Summing up

The following exercise illustrates on one hand the sampling effects with the possible presence of aliasing and on the other hand the effects of truncation with the presence of ripples.

Exercise 3.5 (Effects of sampling and windowing)
Consider the continuous-time signal $x(t) = \exp(-t/t_0)\mathbb{1}(t \in (0, +\infty[)$ with $t_0 > 0$. Its Fourier transform is called $X(F)$.

1. Determine the expression of its Fourier transform $X(F)$. Use it to find the value, as an expression of t_0, of the frequency corresponding to $|X(0)|/\sqrt{2}$.
2. $x(t)$ is sampled at the frequency $F_s = 1/T_s$. Let $x_s(n) = x(nT_s)$ be its sample sequence and $X_s(f)$ the DTFT of $x_s(n)$. Using formula 2.25, find $X_s(f)$ as an expression of $X(F)$. What can you notice?
3. The DTFT is evaluated using only the first M samples $x_s(0)$ to $x_s(M-1)$. What is the resulting effect on the signal's spectrum?
4. Write a program that gives you figure 3.10, illustrating the different signal spectra, continuous-time, discrete-time, and windowed discrete-time ($t_0 = 1/0.7$, $M = 10$, $F_s = 2$ Hz and *Lfft*= 256). Check the results for the period of the ripples.

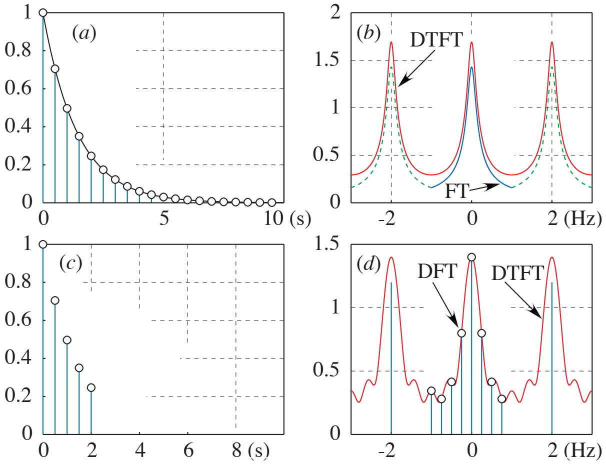


Figure 3.10 – *Effects of sampling and windowing on the signal's spectrum: (a) original signal with its samples, (b) FT et DTFT with aliasing, (c) truncated signal, (d) effects of tuncation (ripples) and values of the DFT.*

3.4 Application examples and exercises

3.4.1 Amplitude modulations

Exercise 3.6 (Amplitude modulation)
Consider a B band, continuous-time real signal $m(t)$, that is to say a signal whose Fourier transform $M(F)$ is equal to zero for $|F| > B$. Let F_0 be a frequency such that $F_0 > B$ (for broadcasting, the order of magnitude for F_0/B is 100).

We call *amplitude modulation* (AM)[1] the operation that generates the signal:

$$x(t) = (1 + km(t))\cos(2\pi F_0 t)$$

F_0 is called the *carrier frequency. k* refers to a positive constant called the *modulation index*, and is chosen so as to have $|km(t)| < 1$. When $|km(t)| > 1$, there is what is called *overmodulation.*

1. Give the expression of the FT $X(F)$ of $x(t)$ as a function of k, $M(F)$ and F_0. How wide is the band occupied by $X(F)$ around F_0?

2. To perform a spectral analysis of the signal $x(t)$, it has to be sampled at $F_s = 500$ kHz. We will assume that $F_0 = 50$ and that $m(t) = \cos(2\pi F_1 t) + 1.8\cos(2\pi F_2 t) + 0.9\cos(2\pi F_3 t)$ where $F_1 = 2{,}310$ Hz, $F_2 = 3{,}750$ Hz and $F_3 = 4{,}960$ Hz. Let $k = 1/2$. Write a program that generates $x(t)$ for a duration of 2 ms. Make sure the chronograms for $m(t)$ and $x(t)$ show no overmodulation.

3. Give the number of samples that have to be processed in order to distinguish the two frequencies contained in the signal.

4. How long must the FFT be if we want a precision of 100 Hz?

5. Write a program that draws the modulated signal's spectrum.

Exercise 3.7 (Carrierless Double Side-Band)
Consider the B band real signal $m(t)$. Its spectrum is called $M(F)$. $M(F) = 0$ for $|F| > B$. Let $F_0 > B$ be a frequency.

The *carrierless amplitude modulation* represented in Figure 3.11, is described by the expression $x(t) = m(t)\cos(2\pi F_0 t)$.

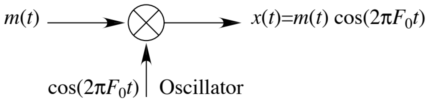


Figure 3.11 – *Carrierless Double Side-Band*

1. Determine the expression of the amplitude spectrum of the modulated signal $x(t)$.

[1] This modulation is called Double Side-Band (DSB) modulation as opposed to the Single Side-Band (SSB) modulation [97].

2. The signal $x(t)$ is modulated a second time by the *local* oscillator $2\cos(2\pi F_0 t + \phi)$. The result is the signal $y(t) = 2x(t)\cos(2\pi F_0 t + \phi)$. Determine, as a function of $M(F)$, F_0 and ϕ, the expression of the spectrum of $y(t)$. Use this result to determine a method for reconstructing the message $m(t)$ from the signal $y(t)$. Why must we have $\phi = 0$? This operation is called *synchronous demodulation* when $\phi = 0$.

3. Consider $m(t) = \cos(2\pi F_1 t) + 1.8\cos(2\pi F_2 t) + 0.9\cos(2\pi F_3 t)$ where $F_1 =$ 2,310 Hz, $F_2 =$ 3,750 Hz and $F_3 =$ 4,960 Hz. Set $F_0 =$ 50 kHz and $F_s =$ 500 kHz as your display frequencies. Write a program that plots the original, modulated and demodulated signals, as well as their spectra.

Exercise 3.8 (Stereophonic signal)

Some frequency modulated broadcasting are sent stereophonically. This means that the received signal makes it possible to reconstruct both the left and right signals. This is achieved by sending the composite signal:

$$c(t) = (l(t) + r(t)) + (l(t) - r(t))\cos(2\pi F_0 t)$$

where $l(t)$ and $r(t)$ refer to the left and right signals respectively. Notice that the signal $(l(t) - r(t))$ is transmitted as carrierless double side-band modulation (see exercise 3.7). For broadcasting, the signals $l(t)$ and $r(t)$ are band-pass signals centered in $F_0 =$ 38 kHz with a bandwidth of 30 kHz.

1. Determine the spectrum's expression for the signal $c(t)$. Draw a quick sketch of its graph.

2. Show that $c(t)$ makes it possible to reconstruct the signal on a monophonic set.

3. Write a program that displays $c(t)$, $2g(t)$ and $2d(t)$ for a sampling frequency of $F_s =$ 1 MHz, where:

   – the signal $l(t)$ is the sum of 5 sines with the amplitudes 0.7, 1.5, 1.9, 2.8 and 3.7, and with the frequencies 380 Hz, 957 Hz, 1,164 Hz, 1,587 Hz and 1,953 Hz respectively;

   – the signal $r(t)$ is the sum of 5 sines with the amplitudes 0.3,1.5, 2.7, 1.7 and 2.3, and with the frequencies 347 Hz, 523 Hz, 1,367 Hz, 2,465 Hz and 3,888 Hz respectively.

   Use this to find a method for separately reconstructing, by sampling, the signals $l(t)$ and $r(t)$.

3.4.2 Frequency modulation

Let $m(t)$ be a B band real signal. The name *frequency modulation* at the carrier frequency $F_0 \gg B$ refers to the operation that generates the signal:

$$x(t) = A\cos(2\pi F_0 t + \Phi(t))$$

where the instantaneous frequency $F_i(t)$ defined by:

$$F_i(t) = F_0 + \frac{1}{2\pi}\frac{d\Phi(t)}{dt} \tag{3.3}$$

is related to $m(t)$ by:

$$F_i(t) = F_0 + \Delta F \times m(t) \tag{3.4}$$

This leads us to $\Phi(t) = 2\pi\Delta F \int_0^t m(u)du$. For commercial broadcasting, $F_0 \gg B$, since $B = 15$ kHz and F_0 belongs to the 88 MHz to 108 MHz band.

We can rewrite $x(t)$ as:

$$x(t) = A\cos(2\pi F_0 t + \Phi(t)) = \text{Re}\left\{Ae^{2j\pi F_0 t + j\Phi(t)}\right\} = \text{Re}\left\{\alpha(t)e^{2j\pi F_0 t}\right\}$$

where $\alpha(t) = Ae^{j\Phi(t)}$. If $F_0 \gg B$, it can be shown [97] that, to obtain the spectrum of $x(t)$, all you have to do is determine the spectrum of $\alpha(t)$, and to translate it, after dividing by 2, around the frequencies $\pm F_0$.

Let us now see the particular case of a sine message $m(t) = \cos(2\pi F_m t)$. In this case, the instantaneous frequency $F_i(t)$ varies between $F_0 - \Delta F$ and $F_0 + \Delta F$. This is why ΔF is called the *frequency deviation*. Let $\beta = \Delta F/B$. β is the *modulation index*[2].

It can be shown that the periodic function $\alpha(t) = A\exp(j\Phi(t)) = A\exp(j\beta\sin(2\pi F_m t))$ has, as its Fourier series expansion:

$$\alpha(t) = A\sum_{n=-\infty}^{+\infty} J_n(\beta)\exp(2j\pi n F_m t)$$

where $J_n(\beta)$ refers to the *Bessel function of the first kind of order n*. Its spectrum shows peaks spaced-out at intervals of F_m. This means that the $x(t)$ spectrum also shows peaks spaced-out at intervals of F_m around $\pm F_0$ (Figure 3.12).

Figure 3.12 was obtained using the `modfm2.m` program. This program plots the spectrum of a signal modulated in frequency by a sine with a frequency of $F_m = 5$ kHz and for a carrier frequency $F_0 = 2$ MHz. Type:

[2] The modulation index plays a fundamental role in communications. In particular, it can be shown the performances of the frequency modulation in the presence of noise increase like β^2.

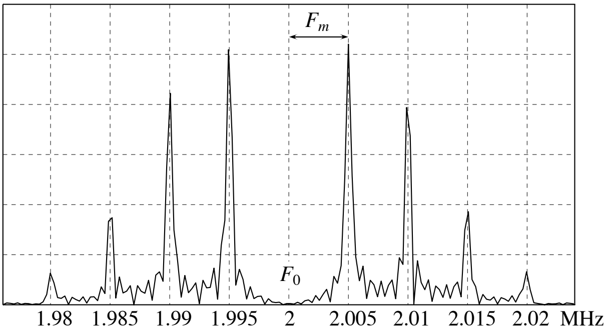


Figure 3.12 – *Spectrum of a frequency modulated sine signal with a modulation index $\beta = 2.4$ (in this case, F_0 peak is missing)*

```
%===== MODFM2.M
Fs=1.0e7;         % Sampling freq. for the simulation
npts=20000; mtime=(0:npts-1)/Fs;
F0=2.0e6;         % Carrier freq.
Fm=5000;          % Signal freq.
disp('Carrier frequency:');
disp(sprintf('\t F0 = %d MHz',F0/1e6));
disp('Message frequency:');
disp(sprintf('\t Fm = %d kHz',Fm/1000));
disp('Instantaneous frequency:');
disp(sprintf('\t fi(t)/(2 pi) = F0 + Deltaf*sin(2*pi*Fm*t)'));
%===== Frequency deviation
disp('Choose the frequency deviation (kHz):');
Df=input(sprintf('\t Deltaf (kHz) = '));
Df=Df*1000;       % Deviation in Hz
beta=Df/Fm;       % Modulation index
theta=2*pi*F0*mtime+beta*cos(2*pi*Fm*mtime); x=cos(theta);
Lfft=32*1024; fq=Fs*(0:Lfft-1)/Lfft;
xf=abs(fft(x,Lfft)); plot(fq,xf); ax=axis; grid
axis([max([0 F0-2*Df]) min([Fs/2 F0+2*Df]) ax(3) ax(4)])
```

The program asks for the value of the frequency deviation (`input...`). By giving, for example, the value `2,4*Fm/1000` (in the program this corresponds to `Deltaf=12`), the result is that the peak at F_0 is erased because in this case, $\beta = 2.4$ and $J_0(2.4) \approx 0$.

This page intentionally left blank

Chapter 4

Linear Filters

When building a model to describe the behavior of some of the most commonly used systems, we often rely on the *superposition principle.* It amounts to assuming *linearity* (the use of Kirchoff's laws are an example). Usually, *time invariance* is also assumed. It consists of saying that, on the time scales that are used, the characteristics of these systems remain unchanged.

Linear filters are defined in the following by these two properties. Because of their importance in the field of signal processing, the next two chapters deal exclusively with filters. This chapter presents the main properties, as well as a few design methods.

4.1 Definitions and properties

Definition 4.1 (Linear filter) *A discrete-time linear filter*[1] *is a system whose output sequence results from the input sequence $\{x(n)\}$ according to the expression:*

$$y(n) = (x \star h)(n) = \sum_{k=-\infty}^{+\infty} x(k)h(n-k) = \sum_{k=-\infty}^{+\infty} h(k)x(n-k) \tag{4.1}$$

where the sequence $\{h(n)\}$ that characterizes the filter is called the impulse response*. The $(x \star h)$ operation is called* convolution *(Figure 4.1).*

For example, the processing defined by $y(n) = \frac{1}{2}x(n) + \frac{1}{2}x(n-1)$ is therefore a linear filtering. The sequence $\{h(n)\}$ is defined by $h(0) = \frac{1}{2}$, $h(1) = \frac{1}{2}$ and $h(n) = 0$ for any value of $n \neq \{0, 1\}$.

For commonly used classes of signals, expression 4.1 is perfectly well defined, and satisfies the linearity property. We will now prove the time invariance property. In order to do this, we will assume that the output sequence $y(n)$

[1] Most of the time, we will just write filter instead of linear filter.

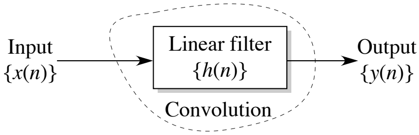


Figure 4.1 – *Discrete-time linear filter*

corresponds to the input signal $x(n)$, and we must determine what output signal $v(n)$ corresponds to the input signal $u(n) = x(n - n_0)$. We can write:

$$\begin{aligned} v(n) &= \sum_{k=-\infty}^{+\infty} u(k)h(n-k) = \sum_{k=-\infty}^{+\infty} x(k-n_0)h(n-k) \\ &= \sum_{p=-\infty}^{+\infty} x(p)h((n-n_0)-p) = y(n-n_0) \end{aligned}$$

However, if there is an n_0 delay for the input, there is also a time delay of n_0 for the output. It should be noted that linear systems as simple as the following two:

$$\begin{aligned} y(n) &= x(n)\cos(2\pi f_0 n) \\ y(n) &= \sum_{k=0}^{+\infty} x(k)h(n-k) \end{aligned}$$

do not possess the time invariance property.

Throughout the rest of this chapter, the two important concepts of *causality* and *BIBO stability* will often be referred to.

Definition 4.2 (Causality) *A system is said to be causal when its output $y(n)$ at time n depends only on the current and previous values of the sequences $x(n)$ and $y(n)$:*

$$y(n) = \mathcal{F}(\{x(p)\}, \{y(p')\}) \text{ with } p, p' \leq n$$

Definition 4.3 (Bounded input – bounded output stability (BIBO)) *A system is said to be BIBO stable[2] if, for any bounded input, the output remains bounded:*

$$\forall n, \ |x(n)| < A \Rightarrow |y(n)| < B$$

We have the following two results:

[2] From now on, when there is no possible confusion, we will just write "stable" instead of "BIBO stable".

Theorem 4.1 *A filter is causal if and only if its impulse response $\{h(n)\}$ is such that:*

$$h(n) = 0 \textit{ when } n < 0 \tag{4.2}$$

HINT: because of 4.1, $y(n)$ depends only on $\{x(n), x(n-1), \dots, x(n-k), \dots\}$ for $k \geq 0$, if and only if $h(k) = 0$ for $k < 0$. If the terms $h(n)$ are equal to zero for $n \geq M$, the *filter memory* is finite, and its value is M. ■

Theorem 4.2 *A linear filter is BIBO stable if and only if its impulse response $\{h(n)\}$ verifies:*

$$\sum_{n \in \mathbb{Z}} |h(n)| < +\infty \tag{4.3}$$

HINT: let us first assume that $\sum_k |h(k)| = M < +\infty$. To any bounded input, that is to say such that $|x(n)| < A$, corresponds a signal $y(n)$ that verifies:

$$|y(n)| \leq \sum_{k \in \mathbb{Z}} |h(k)||x(n-k)| < AM$$

and which is therefore bounded itself. This means that the filter is BIBO stable.

Conversely, we assume that the filter is BIBO stable. We will use proof by contradiction. Let us assume that $\sum_k |h(k)| = \infty$. The question is "can we find at least one bounded input yielding a non-bounded output?". All we have to do is take $x(n) = \text{sign}(h(-n))$ as our bounded input, resulting at the time $n = 0$ in an infinite output $y(0) = \sum_k |h(k)|$. ■

A first use of MATLAB®'s `filter` function

In MATLAB®, the filtering operation is performed by a *built-in* function called `filter`. This function provides us with a *causal* implementation of the filtering operation, so in order to "filter" the input sequence $\{x(1), \dots, x(N)\}$ by the impulse response filter of *finite* length $\{h(1), \dots, h(L)\}$ (the phrase used will be "finite impulse response filter", or FIR filter), we need to type:

```
y = filter(h,1,x);
```

The output sequence $\{y(1), \dots, y(N)\}$ has the same length as the input sequence. To calculate it, the $(L-1)$ values preceding $x(1)$ must first be known. `filter` can accept a fourth argument, used to specify the initial state associated with these values. If this argument is left blank, the `filter` considers that all values are equal to zero, and we have:

$$\begin{aligned}
y(1) &= h(1)x(1) \\
y(2) &= h(1)x(2) + h(2)x(1) \\
&\vdots \\
y(L) &= h(1)x(L) + \cdots + h(L)x(1) \\
&\vdots \\
y(n) &= h(1)x(n) + \cdots + h(L)x(n-L+1) \\
&\vdots \\
y(N) &= h(1)x(N) + \cdots + h(L)x(N-L+1)
\end{aligned}$$

Note that because the impulse response has a finite length, it necessarily verifies 4.3, and therefore the filter is BIBO stable. The fourth argument's purpose will be discussed more in depth in paragraph 5.1.

Example 4.1 (Smoothing filter) We are going to filter the sequence $[0:6]$ with the use of the impulse response filter [1 1]. Type:

```
%===== EXFILTINT.M
clear; x=[0:6]; h=[1 1];
y=filter(h,1,x)
```

The resulting sequence is:

```
y =
       0      1      3      5      7      9     11
```

Example 4.2 (Smoothing filtering of a random sequence)
We will now filter a random sequence `x=rand(50,1)` using a filter with the impulse response:

$$h = \begin{bmatrix} \frac{1}{8} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{8} \end{bmatrix}$$

This filter calculates a weighted mean of five consecutive samples. The result is expected to be less “turbulent” than the signal we started with. Type:

```
%===== EXFILTRAND.M
clear; x=rand(50,1);        % Input signal x
h=[1/8 1/4 1/4 1/4 1/8];   % Impulse response
y=filter(h,1,x);            % Output signal y
plot([x y]); grid
```

The result is shown in Figure 4.2.

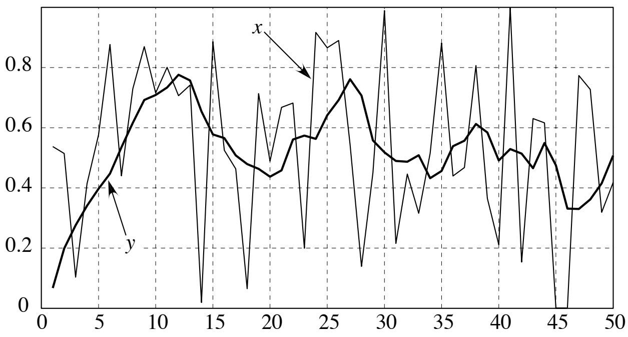


Figure 4.2 – *Filtering a random sequence with the smoothing impulse response filter* $\{1\ 2\ 2\ 2\ 1\}/8$

Definition 4.4 (Step response of a filter)

The step response of a filter is this filter's output when the unit step ($u(n) = 1$ when $n \geq 0$ and 0 otherwise) is fed into the input. The step response's expression is:

$$y(n) = \sum_{k=-\infty}^{n} h(k)$$

In the case of a causal filter, $y(n) = \sum_{k=0}^{n} h(k)$ for $n \geq 0$ and 0 otherwise.

Example 4.3 (Step response of an FIR filter)

Consider the impulse response filter $h(n) = \lambda a^n$ for $0 \leq n \leq 15$ and 0 otherwise. Its step response is denoted $y(n)$. Determine the expression of $y(n)$. Use this to find the expression of λ that leads to 1 for $n \geq 15$. Write a program that calculates, with the `filter` function, the 30 first samples of the step response.

HINT: we have, for $n \leq 15$:

$$y(n) = \sum_{k=0}^{n} h(k) = \lambda \frac{1 - a^{n+1}}{1 - a}$$

For $n \geq 15$, $y(n) = \lambda(1 - a^{16})/(1 - a)$. In order to have $y(n) = 1$ for $n \geq 15$, λ has to be set such that $\lambda = (1 - a)/(1 - a^{16})$. The higher the value of $|a|$, the slower the step response will close in on the final value 1, reaching it at the time $n = 15$. Type the following program:

```
%===== REPINDIC.M
% Impulse responses
N=16; a=[1/2 3/4 7/8]; Nct=length(a);
```

```
a=ones(N,1)*a; hh=(0:N-1)'*ones(1,Nct);
h=a .^ hh; sigm=sum(h);
%===== The impulse responses are normalized in order
%      to compare the rise times
h0=h(:,1)/sigm(1); h1=h(:,2)/sigm(2); h2=h(:,3)/sigm(3);
Lrep=30;                % Response's length
tps=[0:Lrep-1]; x=ones(Lrep,1);
%===== Responses with null initial conditions
y=[filter(h0,1,x) filter(h1,1,x) filter(h2,1,x)];
plot(tps,y,'-',tps,y,'o'); set(gca,'YLim',[0 1.1]); grid
```

The results are given in Figure 4.3. ■

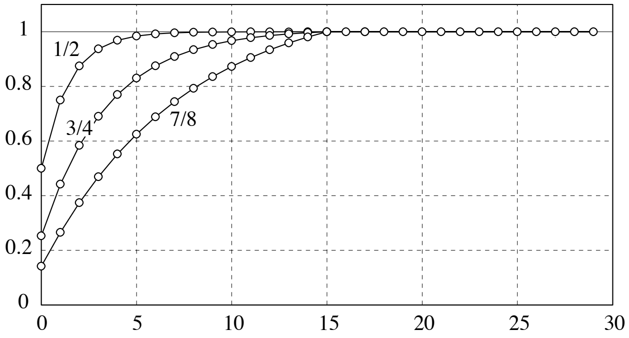


Figure 4.3 – *Step responses*

4.2 The z-transform

An important tool used in discrete-time linear-filtering is the *z-transform* for which we will give the definition, the main properties and a description of how it is used in a filtering context. Most of the properties mentioned in this paragraph are given without proof. Some of them will have no direct use for what follows; however, it is useful simply to know they exist.

4.2.1 Definition and properties

Definition 4.5 (z-transform) *The* z-transform *(ZT) of the sequence* $\{x(n)\}$ *is the function* $X_z(z)$[3] *of the complex variable* z *defined by:*

$$X_z(z) = \sum_{n=-\infty}^{+\infty} x(n)z^{-n} \tag{4.4}$$

[3] When there is no possible confusion, we will use $X(z)$ instead of $X_z(z)$.

for values of z taken inside a ring described by $\{z \in \mathbb{C} : R_1 < |z| < R_2\}$, assumed to be non-empty, and called the convergence area *or* domain of convergence *(Figure 4.4).*

The values of z for which $X_z(z)$ is equal to zero are called zeros, and the values of z for which $X_z(z)$ diverges are called poles.

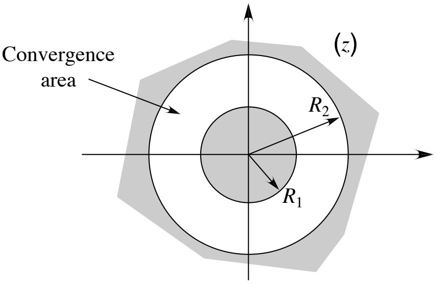


Figure 4.4 – *Convergence area*

The properties enumerated in Appendix A4 provide calculation methods pertaining to the sequence $\{x(n)\}$ or to the function $X_z(z)$.

What should be remembered is that the analytical expression of $X_z(z)$ does not characterize the sequence $\{x(n)\}$. What does characterize the latter is the *pair comprising the function $X_z(z)$ and a convergence area.*

4.2.2 A few examples

The following results can be proved as an exercise, in particular by using the fundamental formula:

$$\frac{1}{1-u} = 1 + u + u^2 + \cdots u^n + \cdots \text{ with } |u| < 1 \tag{4.5}$$

– *Unit impulse*:

$$\delta(n) = \begin{cases} 1 & \text{when } n = 0 \\ 0 & \text{otherwise} \end{cases} \rightarrow \Delta(z) = 1, \forall z \tag{4.6}$$

– *Unit step*:

$$u(n) = \mathbb{1}(n \geq 0) \rightarrow U(z) = \frac{1}{1 - z^{-1}}, \text{ with } |z| > 1 \tag{4.7}$$

– *Ramp*:

$$r(n) = n\mathbb{1}(n \geq 0) \rightarrow R(z) = \frac{z^{-1}}{(1 - z^{-1})^2}, \text{ with } |z| > 1 \tag{4.8}$$

- *Causal exponential*:

$$e_c(n) = a^n \mathbb{1}(n \geq 0) \rightarrow E_c(z) = \frac{1}{(1 - az^{-1})}, \text{ with } |z| > |a| \tag{4.9}$$

- *Anti-causal exponential*:

$$e_a(n) = -a^n \mathbb{1}(n \leq -1) \rightarrow E_a(z) = \frac{1}{(1 - az^{-1})}, \text{ with } |z| < |a| \tag{4.10}$$

We are going to prove 4.8 and 4.10. Because of 14.11 and 4.7, we have:

$$R(z) = -z\frac{dU(z)}{dz} = -\frac{z^{-1}}{(1 - z^{-1})^2}$$

which is expression 4.8.

For 4.10 we have:

$$\begin{aligned} E_a(z) &= -\sum_{n=-\infty}^{-1} a^n z^{-n} = -\sum_{p=0}^{+\infty} a^{-p} z^p + 1 \quad \text{with} \quad |a^{-1}z| < 1 \\ &= 1 - \frac{1}{1 - a^{-1}z} = \frac{1}{(1 - az^{-1})} \end{aligned}$$

Note that $E_c(z)$ and $E_a(z)$ have the same analytical expressions. We can tell them apart by their convergence areas.

The z-transform and the DTFT

When the unit circle belongs to the convergence area, the DTFT exists. In the complex plane, the unit circle can be represented by $z = e^{2j\pi f}$ where f varies from 0 to 1. In that case:

$$X_z(e^{2j\pi f}) = \sum_n x(n) e^{-2j\pi n f}$$

is also the DTFT of $x(n)$ which we denoted by $X(f)$. The unit circle can therefore be scaled, in values of f, from $f = 0$ for $z = 1$, to $f = 1/2$ for $z = -1$, including $f = 1/4$ for $z = j$ (Figure 4.5). To be less specific, if $|z| = 1$:

$$f = \frac{\arg(z)}{2\pi} \tag{4.11}$$

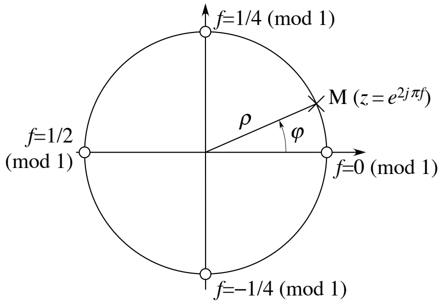


Figure 4.5 – *Unit circle*

4.3 Transforms and linear filtering

Consider again the previous linear filtering. We will refer to the transforms of the sequences $\{x(n)\}$, $\{y(n)\}$ and $\{h(n)\}$ as $X_z(z)$, $Y_z(z)$ and $H_z(z)$ respectively, and to their DTFTs as $X(f)$, $Y(f)$ and $H(f)$ (we will assume that all these functions exist, and in particular that the convergence areas of $X_z(z)$, $Y_z(z)$ and $H_z(z)$ contain the unit circle). We then have:

Property 4.1 (Filtering relations for finite energy signals) *Consider a BIBO stable linear filter with a BIBO stable impulse response $\{h(n)\}$, meaning that $\sum_k |h(k)| < +\infty$. In this case, for finite energy signals $\{x(n)\}$, that is to say such that $\sum_k |x(k)|^2 < +\infty$, we have the following input-output formulas:*

$$y(n) = (x \star h)(n) \rightarrow \begin{cases} Y_z(z) = H_z(z)X_z(z) \\ Y(f) = H(f)X(f) \end{cases} \tag{4.12}$$

$H_z(z)$ is called the filter's *transfer function* (*TF*) and:

$$H(f) = H_z(e^{2j\pi f}) = \sum_{n=-\infty}^{+\infty} h(n)e^{-2j\pi fn} \tag{4.13}$$

is called the *complex gain* or *frequency response.* Remember relation 2.23:

$$h(n) = \int_{-1/2}^{1/2} H(f)e^{2j\pi fn}\, df \tag{4.14}$$

Let $H(f) = G(f)e^{j\phi(f)}$. $G(f) = |H(f)|$ is called the *filter gain*, and $\phi(f) = \arg(H(f))$ is its *phase.*

Property 4.2 *For a linear filter whose impulse response $h(n)$ is real, we have $H_z(z) = H_z^*(z^*)$ and $H(f) = H^*(-f)$. In this case, the gain $G(f) = |H(f)|$ and the real part of $H(f)$ are even functions. Its phase $\phi(f) = \arg(H(f))$ and its imaginary part are odd functions:*

$$G(f) = G(-f) \quad \textit{and} \quad \phi(f) = -\phi(-f)$$

In this case we can restrict the graphical representation of $G(f)$ to $f \in (0, 1/2)$.

Property 4.3 (Harmonic response of a linear filter)
Let $\{h(n)\}$ be the BIBO stable impulse response of a filter, and let $\{x(n) = \exp(2j\pi f_0 n)\}$ be the input signal. The expression of the output signal is:

$$y(n) = H(f_0)x(n)$$

where $H(f) = \sum_n h(n)e^{-2j\pi nf}$ is the DTFT of $\{h(n)\}$.

This can easily be shown by writing:

$$\begin{aligned} y(n) &= \sum_k h(k)x(n-k) = \sum_k h(k)\exp(2j\pi f_0(n-k)) \\ &= \exp(2j\pi f_0 n)\sum_k h(k)\exp(-2j\pi f_0 k) \end{aligned}$$

Complex exponentials are called the *eigenfunctions* of linear filters.

In the particular case of a real filter, that is in the case where $h(n)$ is real, with the input signal $x(n) = \cos(2\pi f_0 n)$, we obtain, by using the expression $x(n) = (\exp(2j\pi f_0 n) + \exp(-2j\pi f_0 n))/2$ and the linearity property:

$$y(n) = \frac{1}{2}H(f_0)\exp(2j\pi f_0 n) + \frac{1}{2}H(-f_0)\exp(-2j\pi f_0 n)$$

Because $h(n)$ is real, $H(f) = H^*(-f)$ and therefore:

$$\begin{aligned} y(n) &= \frac{1}{2}H(f_0)\exp(2j\pi f_0 n) + \frac{1}{2}H^*(f_0)\exp(-2j\pi f_0 n) \\ &= G(f_0)\cos(2\pi f_0 n + \phi(f_0)) \end{aligned}$$

The output sine has the same frequency as the input sine, but its amplitude is multiplied by $G(f_0)$ and its phase is shifted by $\phi(f_0)$.

Consider again the example of the impulse response filter [1 1]. Its transfer function is, for any z:

$$H_z(z) = 1 + z^{-1}$$

Its complex gain can be expressed:

$$H(f) = H_z(e^{2j\pi f}) = 1 + e^{-2j\pi f} = 2e^{-j\pi f}\cos(\pi f)$$

Its gain is therefore $G(f) = 2|\cos(\pi f)|$ and its phase is:

$$\phi(f) = -\pi f \quad \text{if} \quad f \in (-1/2, 1/2)$$

You can check that $G(f) = G(-f)$ and that $\phi(f) = -\phi(-f)$. These characteristics can be directly plotted using the following program:

```
%===== GAINPHASE.M
f=[0:.01:1];
gaincplxe=1+exp(-2*pi*j*f);     % Complex gain
gain=abs(gaincplxe);
phase=angle(gaincplxe)*180/pi; % Phase (degrees)
subplot(211); plot(f,gain); grid
subplot(212); plot(f,phase); grid
```

4.4 Difference equations and rational TF filters

Consider this difference equation (*d.e.*):

$$y_n + a_1 y_{n-1} + \cdots + a_P y_{n-P} = b_0 x_n + b_1 x_{n-1} + \cdots + b_Q x_{n-Q} \tag{4.15}$$

We will assume that the $x(n)$ are known ($n \in \mathbb{Z}$) and that we want to calculate the $y(n)$. Assuming that the z-transforms of $x(n)$ and $y(n)$ exist, consider the z-transforms of the two sides of equation 4.15. Using the delay property, we get:

$$Y_z(z)\left(1 + a_1 z^{-1} + \cdots + a_P z^{-P}\right) = X_z(z)\left(b_0 + b_1 z^{-1} + \cdots + b_Q z^{-Q}\right)$$

which leads us to:

$$H_z(z) = \frac{Y_z(z)}{X_z(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_Q z^{-Q}}{1 + a_1 z^{-1} + \cdots + a_P z^{-P}} = \frac{B(z)}{A(z)} \tag{4.16}$$

The system relating the sequence $y(n)$ to the sequence $x(n)$ is therefore a linear filter with the transfer function $H_z(z)$. This proves that linear recursive equations with constant coefficients perform a linear filtering with rational fractions (ratios of two polynomials) as their transfer functions.

But what convergence area should we choose? This choice is related to the way we calculate the solution to the difference equation.

Example 4.4 (First order difference equation) Consider the example:

$$y(n+1) - ay(n) = x(n)$$

If no further information is given, this equation cannot be solved. We have to indicate the type of solution that we want: causal or non-causal. For these hypotheses we get:

1. Causal solution: a value is set at the origin, for example $y(0)$, and $y(n+1)$ is calculated based on $y(n)$. We write the successive expressions:

$$\begin{aligned} y(1) &= ay(0) + x(0) \\ y(2) &= a^2y(0) + ax(0) + x(1) \\ &\vdots \\ y(n) &= a^ny(0) + a^{n-1}x(0) + \cdots + x(n-1) \end{aligned}$$

2. Anticausal solution: a value is set at the origin, for example $y(0)$, and $y(n)$ is calculated based on $y(n+1)$. We write the successive expressions:

$$\begin{aligned} y(-1) &= y(0)/a - x(-1)/a \\ y(-2) &= y(0)/a^2 - x(-1)/a^2 - x(-2) \\ &\vdots \\ y(-p) &= y(0)/a^p - x(-1)/a^p - \cdots - x(-p) \end{aligned}$$

In an equivalent way, we can define the convergence area of the z-transform of the sequence $\{y(n)\}$ we wish to determine. Depending on whether z is set such that $|z| > \alpha$ or $|z| < \alpha$, where α has to be determined, we get a causal solution or an anti-causal solution, respectively (see properties 4.9 and 4.10).

Example 4.5 (Counterexample: homogeneous first order d.e.) Consider the difference equation:

$$y_n - e^{2j\pi f_0}y_{n-1} = 0 \tag{4.17}$$

By changing over to the z-transform of the two sides, we get $Y_z(z)(1 - e^{2j\pi f_0}z^{-1}) = 0$, from which we infer $Y_z(z) = 0$ and hence $y_n = 0$. However, we can directly check, using the difference equation, that, for any A, the signal $y_n = Ae^{2j\pi f_0 n}$ is solution to (4.17). The fact that this solution cannot be found by changing over to the z-transform of (4.17) is precisely due to the fact that y_n does not have a z-transform.

4.4.1 Stability considerations

We know, first that the possible convergence areas are delimited by poles, and second that stability is ensured so long as the unit circle belongs to the convergence area. Hence, a stable solution exists if and only if there are no poles on the unit circle.

Theorem 4.3 (Stable solution) *A system whose input $x(n)$ and output $y(n)$ obey the recursive equation:*

$$y_n + a_1 y_{n-1} + \cdots + a_P y_{n-P} = b_0 x_n + b_1 x_{n-1} + \cdots + b_Q x_{n-Q}$$

is a stable filter if and only if:

$$A(z) = 1 + a_1 z^{-1} + \cdots + a_P z^{-P} \neq 0 \text{ when } |z| = 1 \tag{4.18}$$

In this case, the impulse response is the sequence $\{h(k)\}$ of coefficients of the Fourier expansion of the rational function $H(f) = B(e^{2j\pi f})/A(e^{2j\pi f})$, where $B(z)$ is defined by 4.16, and we have:

$$y(n) = \textstyle\sum_{k=-\infty}^{+\infty} h(k)x(n-k)$$

Causality

When faced with the recursive equation 4.15, we can always solve it "causally", by writing:

$$y_n = b_0 x_n + b_1 x_{n-1} + \cdots + b_Q x_{n-Q} - (a_1 y_{n-1} + \cdots + a_P y_{n-P})$$

Calculating $y(n)$ requires that we know all the values of $x(k)$ and $y(k)$ for $k \leq n$. Obviously, this can also be inferred from the transfer function's expression. $H_z(z)$, given by 4.16 as a function of z (and not of z^{-1}), is such that the numerator and denominator polynomials are of the same degree, equal to $\max(P, Q)$. Therefore, because of the properties 14.5, a causal sequence $\{h(n)\}$ exists with $H_z(z)$ as its z-transform. This solution corresponds, for $H_z(z)$, to the convergence area $\{z \in \mathbb{C} : |z| > \max_k(|p_k|)\}$. Here the p_k are used to denote the poles of $H_z(z)$, that is the roots of $A(z)$. However, the implementation of this solution is not necessarily stable. It is stable if and only if the convergence area contains the unit circle. This gives the following fundamental result:

Theorem 4.4 (Stable and causal solution)
The system whose input $x(n)$ and output $y(n)$ obey the recursive equation:

$$y_n + a_1 y_{n-1} + \cdots + a_P y_{n-P} = b_0 x_n + b_1 x_{n-1} + \cdots + b_Q x_{n-Q}$$

is a causal and stable linear filter if and only if all of the poles of the transfer function $H_z(z) = B(z)/A(z)$ have a modulus that is strictly less than 1, that is to say if:

$$A(z) = 1 + a_1 z^{-1} + \cdots + a_P z^{-P} \neq 0 \text{ when } |z| \geq 1$$

In that case, the impulse response is such that $h_k = 0$ for $k < 0$ and:

$$y(n) = x(n) + h_1 x(n-1) + \cdots + h_k x(n-k) + \cdots$$

Note the importance of the words *causal* and *stable* when expressing this property. It is quite possible for the system described by the recursive equation 4.15 to have, because of theorem 4.3, a stable solution (the convergence area of $H_z(z)$ contains the unit circle) that is not causal.

4.4.2 FIR and IIR filters

When the polynomial $A(z)$ is only a constant, equation 4.15 can be written:

$$y(n) = b_0x(n) + b_1x(n-1) + \cdots + b_Qx(n-Q)$$

which can be seen, according to 4.1, as the convolution of $x(n)$ with an impulse response that has a *finite* number of non-zero values. The filter is then called a *Finite Impulse Response filter*, or *FIR filter.* A direct consequence is the stability of this type of filter, because of the absence of poles.

When the polynomial $A(z)$ is not a constant, and if the rational function $H_z(z)$ is *irreducible*, the filter is called an *Infinite Impulse Response filter*, *IIR filter*, or *recursive filter.* The use of this term is justified by the fact that the value $y(n)$ is calculated not only using the sampled input values $x(n)$, $x(n-1)\dots$, but also the output ones $y(n-1)$, $y(n-2)\dots$

The expressions “Infinite Impulse Response” and “recursive” are not equivalent. For example, the following filter $H_z(z)$ has a transfer function that can be expressed in two different ways:

$$H_z(z) = 1 + \alpha z^{-1} + \cdots + \alpha^{P-1}z^{-P+1} = \frac{1-\alpha^P z^{-P}}{1-\alpha z^{-1}}$$

This filter is fundamentally FIR. However, its *implementation* can be recursive or non-recursive:

$$\begin{aligned} \text{either } y(n) &= \alpha y(n-1) + x(n) - \alpha^P x(n-P) \text{ (recursive)} \\ \text{or } y(n) &= x(n) + \alpha x(n-1) + \cdots + \alpha^{P-1}x(n-P+1) \text{ (non-recursive)} \end{aligned}$$

Filtering implementation using MATLAB®

The `filter` function, which we have already used for a finite impulse response filter, provides what is called the *causal* solution:

$$\begin{aligned} y(n) &= b_0x(n) + b_1x(n-1) + \cdots + b_Qx(n-Q) \\ &\quad - (a_1y(n-1) + \cdots + a_Py(n-P)) \end{aligned}$$

to the recursive equation 4.15. All you have to do is type:

```
y = filter(B,A,x);
```

where `A=[1  a1 ... aP]` and `B=[b0  b1 ... bQ]`.

The output sequence has the same length as the input sequence. The first term `x(1)` of the sequence `x` represents the oldest element. Theoretically, the calculation requires the Q past values of $\{x(n)\}$ and the P past values of $\{y(n)\}$. Without any further explanation, these values are considered to be equal to zero. The problem of the initial conditions using the fourth parameter of the `filter function` will be discussed more in depth in paragraph 5.1.1.

When using the `filter` *function, because it provides the causal solution to the recursive equation, and because a filter must always be stable, it is imperative that there be no pole with a modulus greater than or equal to* 1. *Hence we must have* $A(z) \neq 0$ *for* $|z| \geq 1$.

4.4.3 Causal solution and initial conditions

Consider this example of a recursive equation to which we want to find a *causal solution*:

$$y(n+2) + a_1 y(n+1) + a_2 y(n) = x(n+1) \text{ with } n \geq 0 \tag{4.19}$$

We can write the successive expressions for $n = 0$, $n = 1\ldots$ and multiply by 1, z^{-1}, $z^{-2}\ldots$:

$$\begin{array}{ccl} 1 & \times & y(2) + a_1 y(1) + a_2 y(0) = x(1) \\ z^{-1} & \times & y(3) + a_1 y(2) + a_2 y(1) = x(2) \\ z^{-2} & \times & y(4) + a_1 y(3) + a_2 y(2) = x(3) \\ \vdots & & \vdots \end{array}$$

By summing these relations, we get:

$$z^2 Y_z(z) - z^2 y(0) - zy(1) + a_1(zY_z(z) - zy(0)) + a_2 Y_z(z) = zX_z(z) - zx(0)$$

Hence the ZT of the output is:

$$Y_z(z) = \frac{z}{z^2 + a_1 z + a_2} X_z(z) + \frac{z^2 y(0) + zy(1) + zy(0) - zx(0)}{z^2 + a_1 z + a_2} \tag{4.20}$$

This relation shows two terms. The first one corresponds to the "forced" part, or the particular solution to equation 4.19. The second term is the "free" part corresponding to the solution to the homogeneous equation which provides the solutions corresponding to the initial conditions.

Generally speaking, if we have:

$$y(n) + a_1 y(n-1) + \cdots + a_N y(n-N) = b_0 x(n) + \cdots + b_M x(n-M) \tag{4.21}$$

with $n \geq M$ where a_k and b_k are constant coefficients, applying the ZT to 4.21, taking into account the initial conditions, leads to expression 4.22:

$$\begin{aligned} \Rightarrow Y_z(z) &= \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} X_z(z) \\ &\quad + \frac{\mathcal{P}_0(z)}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} \end{aligned} \tag{4.22}$$

where $\mathcal{P}_0(z)$ corresponds to the initial conditions (see equation 4.20). The quantity:

$$G_z(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}}$$

is the *transfer function* (*TF*). As you can see, the transfer function coincides with $Y_z(z)/X_z(z)$ when the *initial conditions are equal to zero.*

It can be noted that, because of expression 4.22, simplifying, in the transfer function, an unstable pole of the system by an "unstable" zero (outside the unit disk) of another transfer function does not stabilize the system. There is no reason for applying this simplification to the free part. Consider the following example with $|a| > 1$:

$$X_z(z) \longrightarrow \boxed{\frac{(z-a)B_0(z)}{A_z(z)}} \longrightarrow \boxed{\frac{N_z(z)}{(z-a)D_0(z)}} \longrightarrow Y_z(z)$$

There is indeed a simplification in the transfer function. However, if you consider the free parts of the two systems with the initial conditions polynomials, $\mathcal{P}_0(z)$ and $\mathcal{P}'_0(z)$ respectively, the resulting free part is:

$$\frac{\mathcal{P}_0(z)\mathcal{P}'_0(z)}{(z-a)D_0(z)A_z(z)}$$

which remains unstable.

These considerations can be related to structural concepts (see exercise 12.3 in the following chapter) regarding filters, particularly their observability and controllability, which we will not discuss in this book. Do remember, however, that the transfer function is not enough to characterize the behaviour of a system defined by the recursive equation, because it provides us only with an input-output relation without enough information to lead to a model for the system.

Given the method used to solve 4.19, we can consider that finding a causal solution requires the use of a slightly modified z-transform, called the *unilateral z-transform.*

Definition 4.6 (Causal z-transform)
The causal z-transform (CZT) of the sequence $\{x(n)\}$ is the quantity:

$$X_{zc}(z) = \sum_{n=0}^{+\infty} x(n) z^{-n} \tag{4.23}$$

for values of z such that $R < |z|$.

This is the definition of the z-transform accepted by control engineers who are used to dealing with causal systems, and who take close interest in transient states. Among the properties of the CZT, the lead property is fundamentally different for the two transforms:

Property 4.4 (Time advance)

$$x(n+P) \to z^P X_z(z) - x(0)z^P - x(1)z^{P-1} - \cdots x(P-1)z \tag{4.24}$$

The convergence area is unchanged.

Applying this property to a recursive equation $y(n+P) + a_1 y(n+P-1) + \cdots + a_P y(n) = b_0 x(n+P) + \cdots$ to which we are trying to find a causal solution provides the free part of the answer. Using the $x(n+P) \to z^P X_z(z)$ property only leads to the stationary solution (the one that "started at" $n = -\infty$).

4.4.4 Calculating the responses

Evaluating the impulse response

The `filter` function can be used to directly plot the impulse response of a rational transfer function filter from the coefficients of the recursive equation. Consider as an example the causal and stable filter whose transfer function is:

$$H_z(z) = \frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

with $b_1 = 0.7$, $b_2 = 0.6$, $a_1 = 1.5$, $a_2 = 0.9$ and for which you can check that the two poles are complex, and that each one has a modulus smaller than 1. To plot its causal impulse response, type:

```
%===== REPIMPULS.M
N=60;           % Number of calculation points
b=[1 0.7 0.6];  % Numerator coefficients
a=[1 -1.5 0.9]; % Denominator coefficients
h=filter(b,a,eye(1,N));
stem(h)
```

The input signal is a 1 followed by several 0 (`eye(1,N)`). Obviously this method is not appropriate if the filter, assumed to be stable, is not causal.

Evaluating a complex gain

In order to calculate the complex gain $H_z(e^{2j\pi f})$ of a rational transfer function filter over L points at the frequencies $f = k/L$ with $k \in \{0, \cdots, L-1\}$, we need to calculate the quantities:

$$\left\{ \begin{array}{l} 1 + a_1 e^{-2j\pi k/L} + \cdots + a_P e^{-2j\pi Pk/L} \\ b_0 + b_1 e^{-2j\pi k/L} + \cdots + b_Q e^{-2j\pi Qk/L} \end{array} \right. \qquad k \in \{0, \cdots, L-1\}$$

Notice that they represent the DFTs of the sequences $\{1, a_1, \ldots, a_P\}$ and $\{b_0, b_1, \ldots, b_Q\}$ respectively, over L points. This leads to the following procedure:

The calculation of the complex gain of a filter with $H_z(z) = B(z)/A(z)$ as its transfer function over L points is performed by:

```
H = fft(B,L) ./ fft(A,L);
```

where `B` *and* `A` *represent the sequences of the numerator and denominator coefficients respectively, in decreasing powers of z.*

Exercise 4.1 (The rectangular impulse response filter)
Consider an impulse response $h(n) = 1/M$ when $n \in \{0, \cdots, M-1\}$ and $h(n) = 0$ otherwise.

1. What can be the purpose of such a filter?

2. Calculate the gain and the phase of this filter. What can you say about the latter?

3. Use MATLAB® to plot the gain, the phase, the index response of the filter $h(n)$ for several values of M.

4. Two filters with an impulse response $h(n)$ are arranged in a cascade. What is the gain of the resulting filter?

4.4.5 Stability and the Jury test

Examining a filter's stability requires you to know the poles of the transfer function $H_z(z)$. Using the Jury, or Jury-Lee, test spares you the explicit calculation of their value. It is applied to the denominator $A(z)$ of $H_z(z)$, which we will express as:

$$A(z) = a_0 z^n + a_1 z^{n-1} + \cdots + a_{n-1} z + a_n, \text{ with } a_0 > 0$$

We now set up the following $2n-1$ line array:

$$\begin{array}{|lllllll}
\hline
a_0 & a_1 & a_2 & \cdots & a_i & \cdots & a_n \\
a_n & a_{n-1} & a_{n-2} & \cdots & a_{n-i} & \cdots & a_0 \\
b_0 & b_1 & \cdots & \cdots & \cdots & b_{n-1} & \\
b_{n-1} & b_{n-2} & \cdots & \cdots & \cdots & b_0 & \\
c_0 & c_1 & \cdots & \cdots & \cdots & & \\
c_{n-2} & c_{n-3} & \cdots & \cdots & \cdots & & \\
\cdots & \cdots & \cdots & & & & \\
q_0 & q_1 & q_2 & & & & \\
q_2 & q_1 & q_0 & & & & \\
r_0 & r_1 & & & & & \\
\hline
\end{array}$$

Lines 3 and 4 are filled out based on the first two lines using $b_i = a_0 a_i - a_n a_{n-i}$, then lines 5 and 6 are filled out using $c_i = b_0 b_i - b_{n-1} b_{n-i-1}$, etc. The necessary and sufficient condition for stability is the following set of conditions:

$$\left\{\begin{array}{c} a_0 > |a_n| \\ |b_0| > |b_{n-1}| \\ \vdots \\ |q_0| > |q_2| \\ |r_0| > |r_1| \end{array}\right.$$

and all of them have to be satisfied simultaneously. In practice a $2n-3$ line array must be constructed such that its final values are q_0 q_1 q_2, and the conditions can then be expressed:

$$A(1) > 0, \quad \left\{\begin{array}{ll} A(-1) < 0 & \text{when } n \text{ odd} \\ A(-1) > 0 & \text{when } n \text{ even} \end{array}\right. \quad \text{and} \quad \left\{\begin{array}{c} a_0 > |a_n| \\ |b_0| > |b_{n-1}| \\ \vdots \\ |q_0| > |q_2| \end{array}\right.$$

You can refer to Appendix A5 for the proof.

Exercise 4.2 (Purely recursive first order filter)
Consider the *purely recursive first order* filter with the transfer function:

$$H_z(z) = \frac{1}{1 - az^{-1}}$$

1. This filter is assumed to be stable and causal. Determine the convergence area.

2. By writing $H_z(z)$ as the power series $\sum_{k \in \mathbb{Z}} h(k) z^{-k}$, find the impulse response $h(n)$.

3. Give the expressions of the complex gain, of the gain, and of the phase.

4. Using definition 4.4, determine the expression of the unit step response as a function of a. Determine, as a function of a, the value of λ such that the unit step response tends to 1 when n tends to infinity.

5. Write a program using MATLAB® that calculates, with the use of the `filter` function, the 30 first samples of the index response for $a = -2/3$, $a = 1/2$, $a = 3/4$ and $a = 7/8$. What can you notice?

4.5 Connection between gain and poles/zeros

The positions of the poles and zeros in the complex plane can easily be used to determine the shape of the gain. First of all, we need to study the transfer function of the purely recursive second order filter.

Purely recursive second order filter

Consider the filter whose transfer function coefficients are real:

$$H_z(z) = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{1}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})}$$

We assume that $a_1^2 - 4a_2^2 < 0$, meaning that the two poles are complex conjugates. They are denoted by $p_1 = \rho e^{j\theta}$ and $p_2 = \rho e^{-j\theta}$. We also assume that $p_1 p_2 = \rho^2 = a_2 < 1$ (the product of the roots is equal to a_2). Under these conditions, the poles are inside the unit circle.

Figure 4.7 shows the positions of the poles and the unit circle. The filter is therefore stable and causal, and its transfer function is $H_z(z)$, which will be associated with the convergence area $|z| > \rho$. We will now calculate its impulse response. We need to rewrite the transfer function:

$$H_z(z) = \frac{1}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})} = \frac{1}{p_1 - p_2}\left(\frac{p_1}{1 - p_1 z^{-1}} - \frac{p_2}{1 - p_2 z^{-1}}\right)$$

with $|z| > \rho$. By applying formula 4.5 to the two fractions in z^{-1} of the right side and by isolating the coefficient of z^{-n}, we get for $n \geq 0$:

$$h(n) = \frac{p_1^{n+1} - p_2^{n+1}}{p_1 - p_2} = \rho^n \frac{\sin((n+1)\theta)}{\sin(\theta)}$$

We plotted in Figure 4.6 the impulse response of this filter obtained with the program `repimp2.m`, for $\rho = 0.96$ and $\theta = \pi/12$:

```
%===== REPIMP2.M
N=60;                  % Number of calculation points
tps=[0:N-1];           % Time vector
theta=pi/12; rho=.96; % Parameters
%===== Direct calculation of the response
rep=rho .^ tps .* sin((tps+1)*theta) / sin(theta);
stem(tps,rep); grid
```

The impulse response is a "damped sine".

The farther away the poles are from the unit circle, the faster the impulse response will decrease to zero (it decreases like ρ^n). The filter "forgets" the past faster as ρ is closer to 0. We can therefore define a time after which the memory is "almost completely" gone, by calculating the index n_R such that ρ^{n_R} can be considered to be negligible, meaning that ρ^{n_R} becomes $< \varepsilon$.

Let $\eta = \log(\varepsilon)$, we have:

$$n_R = \frac{\eta}{\log(\rho)} \tag{4.25}$$

This result is common to many situations: we can consider that an IIR filter which, in theory, has an infinite memory of the past, has an "almost finite"

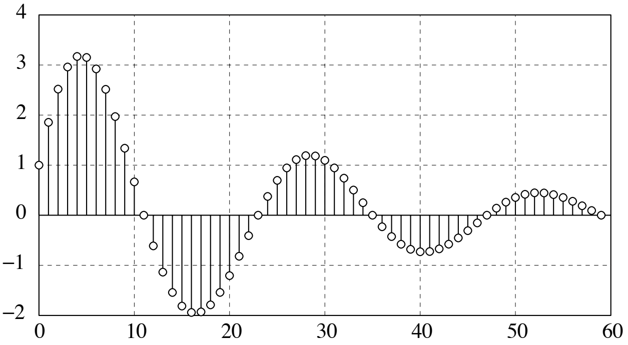


Figure 4.6 – *Impulse response of a second order filter whose two conjugate complex poles have a modulus* $\rho = 0.96$ *and a phase* $\theta = \pi/12$

memory that becomes shorter as the poles move away from the unit circle. This duration corresponds to the time the filter spends “forgetting” the initial conditions, and to get close enough to asymptotic behavior.

Let us now study the gain using the position of the poles. First, notice that the gain:

$$G(f) = \left|H_z(e^{2j\pi f})\right| = \frac{1}{\mathrm{MP1} \times \mathrm{MP2}}$$

where M represents a point on the unit circle with the affix $e^{2j\pi f}$, and P1 and P2 the poles with p_1 and p_2 as their respective affixes. We then see (Figure 4.7) that as MP1 decreases, $G(f)$ increases. Consider, as an example, the case of a denominator:

$$A(z) = 1 - 0.8z^{-1} + 0.4z^{-2}$$

The **purpoles.m** program gives the position of the poles on the complex plane (Figure 4.7).

```
%===== PURPOLES.M
mycircle=exp(2*pi*j*[0:100]/100);
plot(mycircle); hold on
plot(roots([1 -.8 .4]),'x'); hold off; grid
axis('square')
```

The gain plot is represented in Figure 4.8. As an exercise, you can check that, if:

$$-1 < \frac{-a_1(1+a_2)}{4a_2} < 1$$

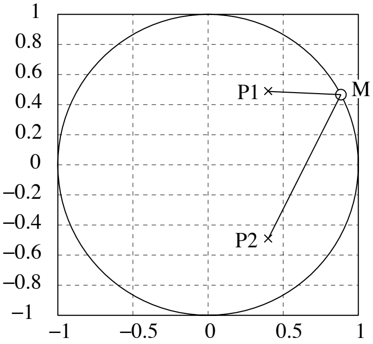


Figure 4.7 – *Position of the poles on the complex plane for a purely recursive second order filter with $a_1 = -0.8$ and $a_2 = 0.4$. We can graphically evaluate the gain as the inverse of the product $MP_1 \times\ MP_2$*

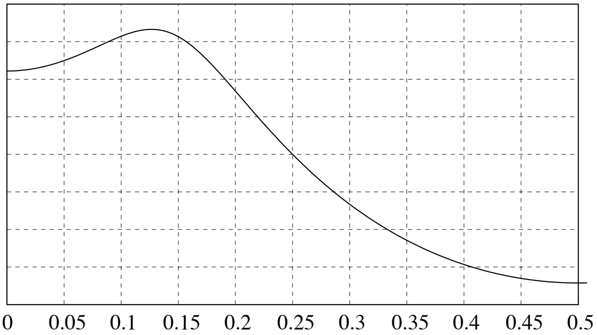


Figure 4.8 – *Gain of a purely recursive second order filter with $a_1 = -0.8$ and $a_2 = 0.4$. Notice the resonant frequency as well as the overvoltage*

the gain shows a maximum at the frequency:

$$f_R = \frac{1}{2\pi} \arccos\left(\frac{-a_1(1+a_2)}{4a_2}\right) \tag{4.26}$$

The frequency f_R is called the *resonant frequency.* The value $G(f_R)$ of the gain at resonant frequency is:

$$G(f_R) = \frac{4a_2}{(1-a_2)^2(4a_2 - a_1^2)}$$

Let the input signal be $x(n) = \varepsilon \cos(2\pi f_R n)$. According to property 4.3, the output signal's expression is $y(n) = G(f_R)\varepsilon \cos(2\pi f_R n + \phi_0)$. If $G(f_R)$ is much greater than 1, the amplitude can reach catastrophic values (like the ones, due to the wind, that caused the suspender cables on the Tacoma bridge to snap, in November 1940, only four months after its inauguration).

Exercise 4.3 (Purely recursive second order)
Consider a *purely recursive second order* filter whose transfer function has real coefficients:

$$H_z(z) = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

1. Sketch the gain in decibels of a second order cell whose poles are given by $p = \rho e^{j\phi}$ for $\rho = 0.9$ and for different values of ϕ.

2. Do the same thing for different values of ρ, ϕ remaining constant.

3. Study the stability, in the sense of "bounded input - bounded output", as a function of a_1 and a_2 by applying the Jury test presented on page 118.

General second order filter

We now add two complex conjugate zeros to the purely recursive second order filter, to see how the frequency response is changed. The transfer function can be expressed as:

$$H_z(z) = \frac{(1 - z_1 z^{-1})(1 - z_2 z^{-1})}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})}$$

By limiting ourselves to the case, which is actually very frequent, of zeros chosen on the unit circle, we get a gain equal to zero at the frequency $f_0 = \alpha/2\pi$ where α refers to the argument of z_1. Such a gain is represented in Figure 4.9. The value of α verifies $2\cos(\alpha) = -1.1$, which leads us to the gain's cancelling frequency $f_0 = \alpha/2\pi \approx 0.3427$. Choosing to put the zero outside of the bandpass reduced the gain's value in the frequency band where the gain was already small.

Example 4.6 (Resonance and rise time)
Consider a *purely recursive second order* filter with the following transfer function:

$$H_z(z) = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

1. The poles are assumed to be complex with a given modulus ρ. Using 4.26, determine a_1 and a_2 such that the resonance frequency is $f_R = 0.1$.

2. Derive the value of the gain G_R at the resonant frequency.

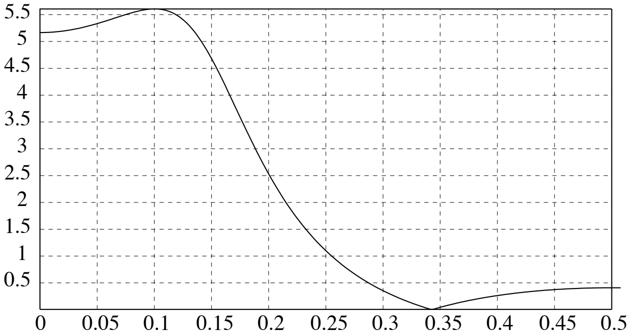


Figure 4.9 – *Gain of a second order filter with two zeros on the unit circle:* $a_1 = -0.8$, $a_2 = 0.4$. $b_1 = 1.1$, $b_2 = b_0 = 1$. *Canceling frequency of the gain* $f_0 \approx 0.3427$

3. Let the input signal be a sine signal $x(n) = G_R^{-1}\sin(2\pi f_R n)$. Determine the output signal's expression.

4. Let the input signal be a "causal sine":

$$x(n) = G_R^{-1}\sin(2\pi f_R n)\mathbb{1}(n \in \mathbb{N})$$

Write a program that plots the output signal $y(n)$ for values of ρ from 0.99 to 0.999. What happens when n tends to infinity?

5. What connection is there between the convergence time and the position of the poles?

HINT:

1. Because the poles are complex $a_2 = \rho^2$. Using 4.26 we obtain:

$$a_1 = -\frac{4a_2}{1+a_2}\cos(2\pi f_R) \tag{4.27}$$

2. The gain is given by $G_R = |1 + a_1 e^{-2\pi j f_R} + a_2 e^{-4\pi j f_R}|^{-1}$.

3. Because of theorem 4.3, if the input signal $x(n) = G_R^{-1}\sin(2\pi f_0 n)$, the output signal's expression is $y(n) = \sin(2\pi f_R n + \phi_R)$ where $\phi_R = \arg(H_z(e^{2j\pi f_R}))$.

4. Type:

```
%===== CTERES.M
clf; clear all; figure(1)
rho=[0.98:0.001:0.999]; fR=0.1; expfR=exp(-2j*pi*fR);
```

```
a2=rho.^2; a1=-4*a2*cos(2*pi*fR) ./(1+a2);
HRm1=1+expfR*a1+expfR^2 *a2;
GRm1=abs(HRm1);phase=angle(HRm1);
N=4000; mtime=(0:N-1); x=sin(2*pi*fR*mtime);   % Input signal
for k=1:length(rho)
    AA=[1 a1(k) a2(k)];
    set(gca,'ylim',[-1.1 1.1],'xlim',[0 30])
    xe=x*GRm1(k); y=filter(1,AA,xe);            % Filtering
    plot(mtime,y); grid; title(sprintf('rho=%5.3f',rho(k)));
    set(gca,'ylim',[-1.1 1.1])
    pause(0.1)
end
```

Notice that when n increases, the filter's output ends up tending to the sine $y(n)$ with the amplitude 1. Everything works as if, after a while, the filter has "forgotten" the initial conditions.

5. It should also be noted that the closer the pole gets to the unit circle, the longer it takes the filter to reach its asymptotic behavior. As we saw on page 120, expression 4.25 makes it possible to evaluate the rise time.

Generally speaking, as the amplitude of the resonance peaks increases, the time constant increases. As a consequence, a small amplitude input can lead to a high amplitude output so long as the energy is provided at the right frequency. This is what happened to the Tacoma bridge because of the wind. ∎

Exercise 4.4 (Suppressing a sinusoidal component)
The rejection problem discussed in this exercise can be solved by using the location of the poles and zeros in the complex plane. We wish to suppress the frequential component $f = f_0$. The first idea that comes to mind is to place a zero on the unit circle at the frequency $f = f_0$. Because we want a real transfer function, we also need the conjugate zero. The numerator can be expressed as:

$$N_z(z) = (1 - e^{2j\pi f_0} z^{-1})(1 - e^{-2j\pi f_0} z^{-1})$$

If we restrict ourselves to this transfer function, the gain is too far from 1 for other frequencies than f_0. This is why a pole is placed close to each zero. Here we are going to impose $\rho e^{2j\pi f_0}$ and $\rho e^{-2j\pi f_0}$ with $\rho < 1$ and ≈ 1. Thus, when $z = e^{2j\pi f}$ is far away from the "pole-zero" pairs, the gain is roughly equal to 1. We have MP $\approx$ MZ and M$\overline{\text{P}}$ $\approx$ M$\overline{\text{Z}}$ (Figure 4.10).

Consider now the second order filter with the transfer function:

$$H_z(z) = H_0 \frac{1 - 2\cos(\phi) z^{-1} + z^{-2}}{1 - 2\rho\cos(\phi) z^{-1} + \rho^2 z^{-2}} \tag{4.28}$$

where $\rho < 1$, $\rho \approx 1$ and H_0 such that $H(1) = 1$.

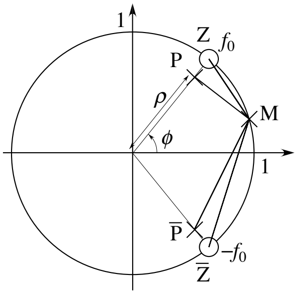


Figure 4.10 – *Graphic interpretation of the gain*

1. Write a program that plots the frequency response of this filter.

2. By making the approximation $2\pi f \approx \phi$ in the neighborhood of the resonant frequency, determine the expression of the frequency interval's width for which the attenuation is higher than 3 dB (decibels).

3. Download (`[x,Fs] = wavread('phrase.wav');`) an audio file sampled at 8000 Hz. Add to the signal a sinusoidal component at 500 Hz.

   Perform the filtering of the signal by using the `filter` function with the filter $H_z(z)$ previously defined. Check that the 500 Hz peak was suppressed by looking at the spectra of the original signal and of the processed signal.

4. Instead of the filter defined by 4.28, let us consider the filter the transfer function of which is the following:

$$H_z(z) = \frac{1}{2}\frac{(1+\rho^2) - 4\rho\cos\varphi z^{-1} + (1+\rho^2)z^{-2}}{1 - 2\rho\cos\varphi z^{-1} + \rho^2 z^{-2}} \tag{4.29}$$

   (a) Verify that the gain is equal to 1 for $f = 0$ and $f = 1/2$.

   (b) Verify that, if the zeros are $e^{\pm j\theta}$:

$$\cos\theta = \frac{2\rho\cos\varphi}{(1+\rho^2)} \tag{4.30}$$

   (c) Write a program that draws the poles and zeros of 4.28 and 4.29 in the complex plane for a few values of ρ.

   (d) Write a program that draws the gain of 4.29 for the same values of ρ.

A description of the gain of a filter

In many practical cases, we have to describe a filter based on its frequency behavior. The frequency band is often partitioned in three zones (see Figure 4.9):

– the *passband* is the frequency band where the gain's values belong to the interval $(1-\delta_p, 1+\delta_p)$, where $\delta_p \gtrsim 0$ is the *passband ripple level*;

– the *stopband* is the frequency band where the gain's values are less than δ_a, where $\delta_a \gtrsim 0$ is the maximum allowed value for the ripples in the stopband;

– the *transition band* is the area where the filter is "moving" between the stopbands and passbands.

Figure 4.11 illustrates these points for a passband filter with characteristics of mediocre quality, whereas Figure 4.12 illustrates the case of a lowpass Butterworth filter (see paragraph 4.7.3).

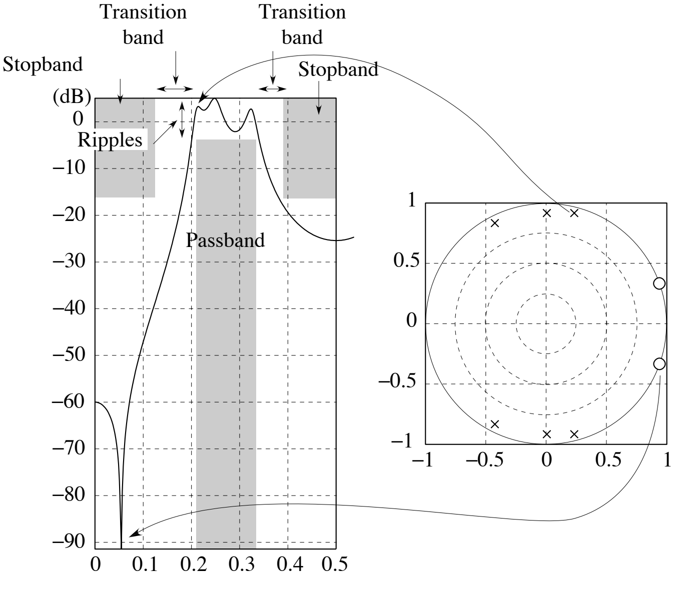


Figure 4.11 – *Specification constraints and positions of the poles and zeros for a passband filter*

Summing up the temporal and spectral aspects of filtering

You will find in this paragraph a certain number of properties that must be kept in mind:

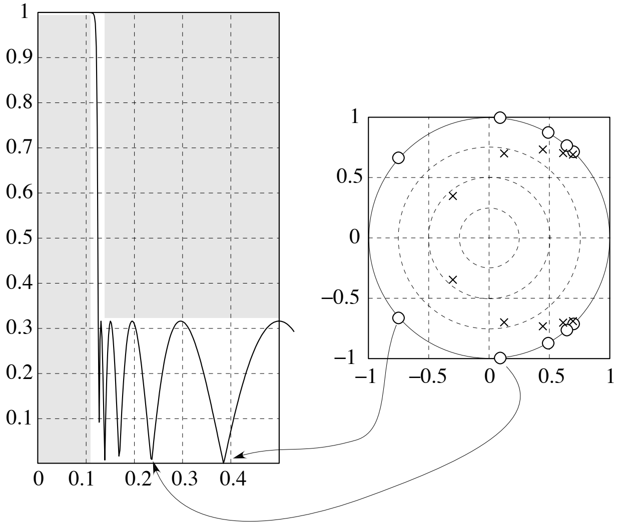


Figure 4.12 – *Specification constraints and positions of the poles and zeros for a low-pass Butterworth filter (paragraph 4.7.3)*

- The impulse response of an FIR filter has a finite length.

- The impulse response of an IIR filter is the sum of "damped sinusoids" (when there are complex poles) that decrease exponentially, decreasing faster as the poles are closer to the unit circle. The decrease duration, or time constant, has the same order of magnitude as $1/\log(\rho_M)$ where ρ_M denotes the modulus of the pole with the highest modulus.

- A pole very close to the unit circle means an important gain at the *resonant frequency*, which is roughly equal to $\arg(p)/2\pi$.

- The angular part of the complex plane where the poles can be found corresponds to the bandwidth. Depending on where the poles are, the filter can be *low-pass*, *high-pass*, or *band pass*. These names mean that the filter allows low or high frequencies, or a given band of frequency components respectively, in the input signals to pass.

- The higher the number of poles, the more the bandwidth ripples can be attenuated. A simplified way of seeing it is to imagine that as z travels along the unit circle, the overvoltages associated with each pole do not have the time to dampen.

- The angular part of the complex plane where the zeros can be found correspond to the stopband.

- The higher the number of zeros, the more the stopband ripples can be attenuated.
- In the case where there are zeros on the unit circle, the gain is equal to zero at the corresponding frequencies.
- A low-pass filter reduces the high frequency components making the transitions in the temporal domain smoother.

4.6 Minimum phase filters

Minimum phase filters have some optimality properties, particularly in terms of response time. Before studying these properties the reader must first be introduced to the concept of all-pass filters.

Definition 4.7 (All-pass filter) *An* all-pass filter *is a stable filter with a gain equal to* 1.

Theorem 4.5 *Let $\{b_k\}$ be a sequence of N complex values with their moduli smaller than* 1. *This means that the filter whose transfer function is:*

$$P_z(z) = \prod_{k=1}^{N} \frac{z^{-1} - b_k^*}{1 - b_k z^{-1}} = \prod_{k=1}^{N} \frac{1 - b_k^* z}{z - b_k} \quad \textit{with } |z| > \max_k |b_k| \tag{4.31}$$

is an all-pass, stable, causal filter and verifies:

$$|P_z(z)| \begin{cases} < 1 & \textit{if} \quad |z| > 1 \\ = 1 & \textit{if} \quad |z| = 1 \\ > 1 & \textit{if} \quad |z| < 1 \end{cases}$$

All you have to do is check it for the term $P_k(z) = (1 - b_k^* z)/(z - b_k)$. First, by taking $z = e^{2j\pi f}$, we get:

$$\frac{1 - b_k^* e^{2j\pi f}}{e^{2j\pi f} - b_k} = e^{2j\pi f} \frac{e^{-2j\pi f} - b_k^*}{e^{2j\pi f} - b_k}$$

the modulus of which is 1 (the modulus of the ratio of two complex conjugate numbers is equal to 1). For $|z| < 1$, notice that $|P_k(0)| = 1/|b_k| > 1$. This means we necessarily have $|P_k(z)| > 1$ for $|z| < 1$, otherwise this would contradict the maximum theorem [20] for holomorphic functions. When $|z| > 1$, the same argument is used after noticing that $|P_k(1/z^*)| = 1/|P_k(z)|$.

Moreover, because the poles are strictly inside the unit circle, the filter is stable and causal.

How are the zeros and poles of $H_z(z)$ placed? Again, all we have to do is limit ourselves to the term $(z^{-1} - b_k^*)/(1 - b_k z^{-1})$. The pole is in $p_k = b_k$

and the zero is in $z_k = 1/b_k^*$. The moduli of the complex values p_k and z_k are therefore the inverse of one another, and their phases are the same. In the complex plane, the two points are therefore the transforms of one another by the inversion centered in O with a ratio of 1 (Figure 4.13).

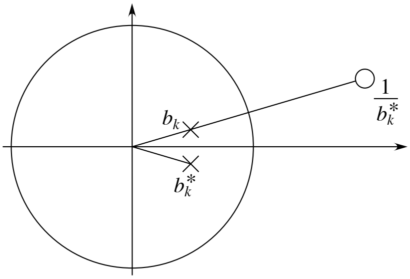


Figure 4.13 – *Positions of poles and zeros in an all-pass filter*

Exercise 4.5 (All-pass filter, properties of the maximum)
Prove geometrically, using the properties of the inversion, the property stated in theorem 4.5.

Exercise 4.6 (All-pass filter)
Consider an all-pass filter. The input and output sequences will be denoted by $x(n)$ and $y(n)$ respectively.

1. Show that $\sum_{n=-\infty}^{+\infty} |x(n)|^2 = \sum_{n=-\infty}^{+\infty} |y(n)|^2$.

2. Show that an all-pass *causal* filter verifies, for any N, $\sum_{n=-\infty}^{N} |x(n)|^2 \geq \sum_{n=-\infty}^{N} |y(n)|^2$.

Theorem 4.6 *Let $H_z(z)$ be the rational transfer function of a stable and causal filter. If we transform, by the inversion centered in O with a ratio of 1, the position of any zero, we get a stable, causal filter with the same gain.*

If we denote by z_1 a zero of $H_z(z)$ and consider the transfer function:

$$F_z(z) = H_z(z)\frac{z^{-1} - z_1^*}{1 - z_1 z^{-1}}$$

Because the poles have not moved, the filter whose transfer function is $F_z(z)$ is stable and causal. However, the numerator has changed, but according to 4.5:

$$\left|F_z(e^{2j\pi f})\right| = \left|H_z(e^{2j\pi f})\right|$$

To put it simply, if $|z_1| > 1$, we made the zero go from outside the unit circle to inside it, and without changing the filter's gain. If we assume that $H_z(z)$ has Q zeros, then there are 2^Q ways of placing them, either outside or inside the unit circle. All the resulting filters have the same gain, but different phases.

One of them, the filter with all its zeros inside the unit circle, plays an important role. It is called the *minimum phase filter.*

Definition 4.8 (Minimum phase) *A stable and causal filter is called a minimum phase filter if all the zeros of its transfer function, assumed to be rational, are strictly inside the unit circle. Notice that the inverse filter is also a stable, causal and minimum phase filter.*

Often when designing a rational filter with Q zeros and P poles, the only information given is the frequency gain. It is implicit that the filter is stable, causal and minimum phase, making the solution unique.

Exercise 4.7 (Minimum phase filter)
If the numerator of the rational function $B(z)/A(z)$ has Q zeros, there are 2^Q ways of placing them, either inside or outside the unit circle, without changing the modulus of $B(e^{2j\pi f})/A(e^{2j\pi f})$. If $A(z) \neq 0$ for $|z| \geq 1$, the impulse responses associated to all these rational functions are causal and stable. In this exercise, the index m refers to the causal, stable, *minimum phase* filter (all the zeros are inside the unit circle), whereas the absence of an index refers to one of the $(2^Q - 1)$ other causal and stable filters $G(f)$ with the same gain $|G(f)| = |G_m(f)|$. The responses of the filters $G_m(f)$ and $G(f)$ to the signal $x(n)$ are denoted $y_m(n)$ and $y(n)$ respectively.

1. Show that if $x(n)$ is causal, then $|y_m(0)| \geq |y(0)|$.

2. Show that $\sum_{n=-\infty}^{N} |y_m(n)|^2 \geq \sum_{n=-\infty}^{N} |y(n)|^2$.

The results of exercise 4.7 show that, among all the causal and stable filters with the same gain, the minimum phase filter is the one with the fastest response.

Definition 4.9 (Group time, phase time)
The phase delay at f_0 is the quantity (its dimension is time) given by:

$$\tau_\phi(f_0) = -\frac{1}{2\pi} \left.\frac{\Phi(f)}{f}\right|_{f=f_0}$$

The group delay in f_0 is defined by:

$$\tau_g(f_0) = -\frac{1}{2\pi} \left.\frac{d\Phi(f)}{df}\right|_{f=f_0}$$

Among all the filters with the same gain, the minimum phase filter is the one with the lowest phase delay and the lowest group delay.

As an example we will demonstrate the group delay property: since the zero $b = |b|e^{2j\pi f_b}$ contributes the factor $1 - bz^{-1}$ to the transfer function, the corresponding phase contribution is $\phi_b(f) = \arg(1 - be^{-2j\pi f})$. Then $\phi_b(f)$ contributes the following to the group delay:

$$-\frac{1}{2\pi}\frac{d\phi_b(f)}{df} = \frac{|b| - \cos(2\pi(f - f_b))}{|b| + |b|^{-1} - 2\cos(2\pi(f - f_b))} \tag{4.32}$$

The denominator and f_b are invariant by reflecting the zero b outside of the unit circle. However, by reflecting b outside of the unit circle, the magnitude of $|b|$ in the numerator of (4.32) is increased. Thus, having b inside the unit circle minimizes the group delay contributed by the factor $(1 - bz^{-1})$. We can extend this result to the general case of more than one zero since the phase of the multiplicative factors of the form $(1 - b_i z^{-1})$ is additive.

The following example gives an explanation for the names group delay and phase delay.

Example 4.7 (Group delay, phase delay)
Consider the complex signal $x(n) = m(n)\exp(2j\pi f_0 n)$. $x(n)$ can be seen as a sine with the frequency f_0 and its amplitude modulated by $m(n)$. This signal is the input for a complex gain filter $\exp(j\Phi(f))$.

We assume that the frequency f_0 is greater than the bandwidth B of the signal $m(n)$. Hence $X(f)$ fills up a very narrow B frequency band around f_0. It is then justified to approximate Φ by its first order series expansion around f_0. By assuming that τ_g is an integer (or the closest integer), determine the output signal's expression as a function of $m(n)$, H_0, τ_g, τ_ϕ and f_0.

HINT: with $\Phi(f) \approx \Phi(f_0) + (f - f_0)\Phi'(f_0)$ and definition 4.9, we have:

$$\begin{aligned} H(f) &= H_0\exp(j\Phi(f)) \approx H_0\exp(j\Phi(f_0) + j(f - f_0)\Phi'(f_0)) \\ &= H_0\exp(-2j\pi f_0\tau_\phi)\exp(-2j\pi\tau_g(f - f_0)) \end{aligned}$$

We also have $X(f) = M(f - f_0)$, hence:

$$Y(f) = H_0\exp(-2j\pi f_0\tau_\phi)\exp(-2j\pi\tau_g(f - f_0))M(f - f_0)$$

According to the delay property, the term $T(f) = \exp(-2j\pi\tau_g f)M(f)$ is the DTFT of the sequence $m(n - \tau_g)$. Therefore, the term $T(f - f_0)$ is the DTFT of the sequence $m(n - \tau_g)\exp(2j\pi f_0 n)$. If we multiply by the term $H_0\exp(-2j\pi f_0\tau_\phi)$ which is independent from f, we get:

$$\begin{aligned} y(n) &= H_0\exp(-2j\pi f_0\tau_\phi)m(n - \tau_g)\exp(2j\pi f_0 n) \\ &= H_0 m(n - \tau_g)\exp(2j\pi f_0(n - \tau_\phi)) \end{aligned}$$

In the end, we have:

$$m(n)\exp(2j\pi f_0 n) \longrightarrow H_0 m(n-\tau_g)\exp(2j\pi f_0(n-\tau_\phi))$$

The envelope is, on the whole, delayed by τ_g, hence the name group delay, and the phase of the carrier is shifted by τ_ϕ. This result can easily be extended to the signal $x(n) = m(n)\cos(2\pi f_0 n)$. All we need to do is decompose the cosine as two exponentials, one around $-f_0$, and the other around $+f_0$, ■

4.7 Filter design methods

The methods explained in this paragraph make it relatively easy to design the most common filters. We will only be using the *window method* and the methods taken from "discrete-time" to "continuous-time" transformations.

The first paragraph shows the relation between the gain of a given continuous-time filter and the gain of the digital filter that implements it.

4.7.1 Going from the continuous-time filter to the discrete-time filter

Consider a filter whose impulse response is $h(t)$ with the continuous-time input signal $x(t)$. The output signal is denoted by $y(t)$. The Fourier transforms of $x(t)$, $y(t)$ and $h(t)$ are denoted by $X(F)$, $Y(F)$ and $H(F)$ respectively.

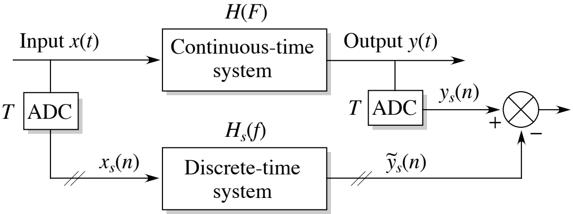


Figure 4.14 – *Comparing the outputs at sampling times*

Consider $H(F)$. We are going to try and find a discrete-time filter with an impulse response $h_s(n)$, which would have the samples $y_s(n) = y(nT)$ of the signal $y(t)$ as its output when it has the samples $x_s(n) = x(nT)$ as its input. In order to do this, we are going to calculate on one hand the DTFT of the output samples of the digital filter's output, and on the other hand the DTFT associated with the output samples of the filter. By making these two expressions equal, we obtain a relation between the two filters.

The discrete-time filter output samples we are trying to determine will be denoted $\tilde{y}_s(n)$. Using obvious notations, the DTFT of $\tilde{y}_s(n)$ is given by:

$$\tilde{Y}_s(f) = H_s(f)X_s(f) \tag{4.33}$$

Because, by definition, $H_s(f)$ is periodic with period 1, the function $\widetilde{H}(f)$, defined on $(-1/2,+1/2)$ by $\widetilde{H}(f) = H_s(f)\mathbb{1}(f \in [-1/2, 1/2])$, is such that:

$$H_s(f) = \sum_k \widetilde{H}(f-k)$$

To put it more graphically, $\widetilde{H}(f)$ represents the truncated pattern of the function $H_s(f)$ in the $(-1/2, 1/2)$ band. Using formula 2.25 (see page 70), which gives us $X_s(f)$ as a function of $X(F)$, 4.33 can also be written:

$$\tilde{Y}_s(f) = \sum_k \widetilde{H}(f-k) \times \frac{1}{T}\sum_k X((f-k)F_s) \tag{4.34}$$

If we now assume that $x(t)$ is $(-F_s/2, +F_s/2)$ band limited, which happens in practice when an anti-aliasing filter is used before the analog-to-digital converter, $X(F) = \sum_k X(F - kF_s)\mathbb{1}(F \in (-F_s/2, +F_s/2))$. In this case, the truncated pattern of $\sum_k X(F - kF_s)$, in the $(-F_s/2, +F_s/2)$ band, coincides with $X(F)$ and expression 4.34 can also be written:

$$\tilde{Y}_s(f) = \frac{1}{T}\sum_k \widetilde{H}(f-k)X((f-k)F_s) \tag{4.35}$$

As for continuous-time, we have $Y(F) = H(F)X(F)$, and therefore, by using once again formula 2.25, the DTFT of the sequence $y_s(n) = y(nT)$ can be expressed:

$$Y_s(f) = \frac{1}{T}\sum_k Y((f-k)F_s) = \frac{1}{T}\sum_k H((f-k)F_s)X((f-k)F_s) \tag{4.36}$$

In order for 4.35 and 4.36 to coincide, we need:

$$\widetilde{H}(f) = H(fF_s)\mathbb{1}(f \in (-1/2, 1/2))$$

Hence the method for constructing the gain $H_s(f)$ from $H(F)$:

– $H(F)$ is truncated at the interval $(-F_s/2, +F_s/2)$.

– The frequency scale is normalized by dividing by F_s.

– The resulting function is periodized[4] with period 1.

Once the function $H_s(f)$ has been determined, creating and implementing it requires certain techniques, some of which are given in this chapter (see exercise 4.11). Example 4.8 shows an application for which the filtering is applied directly to the frequency.

[4] If the resulting function shows jumps such that $X(f_0^-) = a^-$ and $X(f_0^+) = a^+$, the condition $X(f_0) = (a^- + a^+)/2$ is set for continuity reasons.

Example 4.8 (Analytical signal)
As a reminder (see example 1.1), the analytical signal $z(t)$ associated with the continuous-time real signal $x(t)$, is obtained by filtering $x(t)$ using the filter with $2U(F)$ as its complex gain, where $U(F)$ is the unit-step function, equal to 1 if $F > 0$ and 0 if $F < 0$.

We also saw in the same example that $x(t)$ was the real part of $z(t)$ and that the Hilbert transform of $x(t)$ was defined as the imaginary part of $z(t)$:

1. Using $Z(F) = 2U(F)X(F)$ as the frequency's expression, find the discrete-time filtering that creates the samples of $z(t)$ by working with the samples of the signal $x(t)$.

2. We want to perform the frequency filtering $Z(F) = 2U(F)X(F)$, using the DFT. What problems are we going to be faced with?

3. Write a program that calculates the analytical signal of the real signal $x(n)$ resulting from the sampling of $x(t)$. Name this program `siganal.m`.

4. Use the function created for plotting the impulse response of the Hilbert transform response, by typing `siganal([zeros(32,1);eye(1,32)])`.

5. Record a speech signal, sampled at $F_s = 8{,}000$ Hz. Using the previous function, calculate its Hilbert transform. Visualize the signal, then listen to it.

HINT:

1. The analytical signal is obtained by applying the filter with the gain $H(F) = 2U(F)$ to the real signal $x(t)$. Because we are working with the sampled signals, the filter's gain is, in the interval $(-1/2, +1/2)$:

$$H_s(f) = \begin{cases} 0 & \text{when} \quad -1/2 < f < 0 \\ 1 & \text{when} \quad f \in \{0, 1/2\} \\ 2 & \text{when} \quad 0 < f < 1/2 \end{cases}$$

   The rest of the function $H_s(f)$ is obtained by periodizing the expression above with period 1.

2. In the expression $Y_s(f) = H_s(f)X_s(f)$, substituting the DFTs for DTFTs leads to replacing a linear convolution with a circular convolution. In practice, if the signal block on which the DFT is calculated is much greater than the duration of the filter's impulse response, the resulting error is small, except at the beginning and at the end of the block. This implementation can therefore be used since the impulse response of the analytical filter decreases like $1/n$.

3. We are going to perform an L length DFT on the considered block signal. As $x(n)$ is the real part of the analytical signal $z(n)$, as in the continuous case, we have to verify $H_{2U}(k)+H^*_{2U}(-k \bmod L)=2$ for $k=0$ to $L-1$. We then multiply by 2 the portion of the DFT that goes from the indices 2 to $L/2$ (positive frequencies), by 0 the portion of the DFT that goes from the indices $L/2+2$ to L (negative frequencies), and finally by 1 the terms for the indices 1 (zero frequency) and $L/2+1$ (frequency 1/2). Type:

```
function sa=siganal(x)
%%====================================================%
%% Calculating the analytical signal of a real signal %
%% Synopsis: sa=SIGNANAL(x)                           %
%%     x = real signal                                %
%%    sa = analytical signal associated with x        %
%%====================================================%
x=x(:); N=length(x);
xf=fft(real(x));
if rem(N,2)==0
    twoUf=[1; 2*ones(N/2-1,1); 1; zeros(N/2-1,1)];
else
    twoUf=[1; 2*ones((N-1)/2,1); zeros((N-1)/2,1)];
end
saf=xf .* twoUf; sa=ifft(saf);
return
```

4. We can now check the $1/n$ decrease of the Hilbert transform filter's impulse response. Type:

```
%===== RIHILBERT.M
clear; L=32;
%===== Impulse response of the analytical filter
riA=siganal([zeros(L,1);eye(L,1)]);
%===== Impulse response of the Hilbert filter
riH=imag(riA);
hyperbola=zeros(L,1); hyperbola(2:2:L+1)=(2/pi) ./ (1:2:L);
hyperbola=[0;-hyperbola(L:-1:2);hyperbola];
stem(riH,'x'); hold on; plot(hyperbola,'r:'); hold off
```

The imaginary part of the result is assumed to be equal to the Hilbert transform. A direct calculation of the inverse DTFT of $-j\text{sign}(f)$ leads us to $2/n\pi$ if n is odd, and 0 otherwise.

5. Type:

```
%===== HILPHRASE.M
clear; load phrase
```

```
yhilb=imag(siganal(y));
subplot(211), plot(y); grid
subplot(212), plot(yhilb); grid
soundsc(yhilb,8000);
```

By looking at the graph of the signal resulting from the Hilbert transform, you can notice important modifications compared with the original signal, even though the signal remains perfectly clear when listened to. This is sometimes explained by saying that the human ear is mainly sensitive to the Fourier transform's modulus rather than to its phase, and it just so happens that the Hilbert transform filter's gain is equal to 1, hence the input and output Fourier transforms have the same modulus. ■

4.7.2 FIR filter design using the window method

The *window method* allows us to design finite impulse response filters, based on an ideal frequency response. This implementation always leads to unwanted ripples in the frequency response. Furthermore, the calculation time during a filtering operation, expressed as a number of *MAC* operations (see footnote on page 77), is usually much greater than for an equivalent IIR structure.

Its main advantage is that the calculation of the coefficients is simple. Other, more complex algorithms (Remez)[9], use optimization criteria, such as a separate setting for passband and stopband ripples.

We will start with an example.

Linear phase FIR filter

Most of the time, the *window method* is used to satisfy the *linear phase condition*, which merely corresponds to a delay (see property 14.2 of the TFTD). For "audio" applications, this property is often required to make sure the filtered signal maintains a certain level of quality. It is related to the coefficient symmetry property. To understand this, consider the impulse response $\{h(n)\}$ such that $h(n) = h(P-n)$ for $n \in \{0, \ldots, P\}$ and $h(n) = 0$ otherwise. We then have:

$$\begin{aligned} H(f) &= h(0) + h(1)e^{-2j\pi f} + \cdots \\ &\qquad + h(P-1)e^{-2j\pi(P-1)f} + h(P)e^{-2j\pi Pf} \\ &= 2e^{-j\pi Pf}\left(h(0)\cos(\pi Pf) + h(1)\cos(\pi(P-2)f) + \cdots\right) \end{aligned}$$

This expression shows that the phase of $H(f)$ is $\Phi(f) = \pi - P\pi f$ or $\Phi(f) = -P\pi f$ depending on the sign of the real term between parentheses. The filter is then called a linear-phase filter. It can easily be checked that taking $h(n) = -h(P-n)$ leads to a similar result.

In practice, in order to satisfy the symmetry properties, there are two possibilities, depending on whether the impulse response we are trying to determine has an odd or an even number of coefficients.

We will illustrate this with a simple example: consider a filter we want to create, the gain of which is represented in Figure 4.15, and called a *half-band filter* (notice that $H(f)e^{-\pi j f}$ also obeys this property).

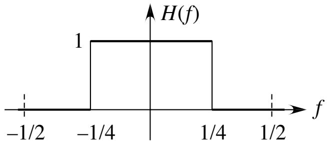


Figure 4.15 – *Example: half-band filter*

1. If N is odd, the coefficients are given by:

$$h(n) = \int_{-1/2}^{1/2} H(f)e^{2j\pi n f}\,df = \int_{-1/4}^{1/4} e^{2j\pi n f}\,df = \frac{\sin(n\pi/2)}{n\pi}$$

The moduli of the terms of this *infinite length* sequence decrease with n. Keeping only N terms introduces an error in the filter's output signal. In our example with $N = 15$, there is some justification for keeping only the terms for the indices from $n = -7$ to $n = +7$. Hence this filter is non-causal. When performing real-time filtering, the impulse response has to be delayed by 7 samples to ensure causality. This leads to an equivalent output delay of $(N-1)/2$ samples.

Causal implementation gives the finite impulse response:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$h(n)$	$-\frac{1}{7\pi}$	0	$\frac{1}{5\pi}$	0	$-\frac{1}{3\pi}$	0	$\frac{1}{\pi}$	$\frac{1}{2}$	$\frac{1}{\pi}$	0	$-\frac{1}{3\pi}$	0	$\frac{1}{5\pi}$	0	$-\frac{1}{7\pi}$

The filter's complex gain is:

$$\begin{aligned} H(f) &= -\frac{1}{7\pi} + \frac{1}{5\pi}e^{-4j\pi f} - \frac{1}{3\pi}e^{-8j\pi f} + \frac{1}{\pi}e^{-12j\pi f} + \frac{1}{2}e^{-14j\pi f} \\ &\quad + \frac{1}{\pi}e^{-16j\pi f} - \frac{1}{3\pi}e^{-20j\pi f} + \frac{1}{5\pi}e^{-24j\pi f} - \frac{1}{7\pi}e^{-28j\pi f} \\ &= e^{-14j\pi f}\left(-\frac{2}{7\pi}\cos(14\pi f) + \frac{2}{5\pi}\cos(10\pi f)\right. \\ &\quad \left. -\frac{2}{3\pi}\cos(6\pi f) + \frac{2}{\pi}\cos(2\pi f) + \frac{1}{2}\right) \end{aligned}$$

Its phase is linear, and given by $\Phi(f) = \Delta - 14\pi f$, where Δ equals 0 or π, depending on the sign of the term between parentheses.

2. If N is odd (see Figure 4.16), the coefficients are given by:

$$\begin{aligned} h(n) &= \int_{-1/2}^{1/2} \left[H(f)e^{-j\pi f}\right] e^{2j\pi n f} df = \int_{-1/4}^{1/4} e^{-j\pi f} e^{2j\pi n f} df \\ &= \frac{\sin((2n-1)\pi/4)}{(2n-1)\pi/2} \end{aligned}$$

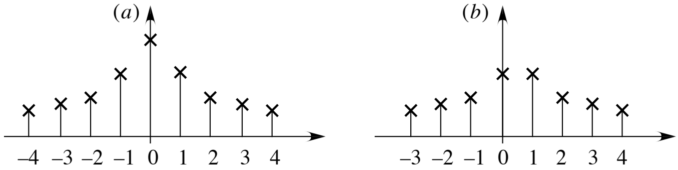


Figure 4.16 – *Half-band filter: comparison of the impulse responses in the odd case (a) and in the even case (b)*

With $N = 6$, the indices n from -2 to $+3$ are kept. The causal implementation consists of designing the finite impulse response filter:

n	0	1	2	3	4	5
$h(n)$	$-\frac{\sqrt{2}}{5\pi}$	$\frac{\sqrt{2}}{3\pi}$	$\frac{\sqrt{2}}{\pi}$	$\frac{\sqrt{2}}{\pi}$	$\frac{\sqrt{2}}{3\pi}$	$-\frac{\sqrt{2}}{5\pi}$

This linear phase filter has the complex gain:

$$H(f) = \frac{2\sqrt{2}}{\pi} e^{-5j\pi f} \left(-\frac{1}{5}\cos(5\pi f) + \frac{1}{3}\cos(3\pi f) + \cos(\pi f) \right)$$

Type:

```
%===== EVENODD.M
Lfft=512; freq=[0:Lfft-1]' / Lfft;
%===== Odd case
N=11; K=(N-1)/2; idx=(-K:K);
hi=sin(idx*pi/2) ./ idx / pi; hi(K+1)=.5; hi=hi/sum(hi);
Hif=abs(fft(hi,Lfft));
%===== Even case
N=12; K=N/2; idx=2*(-K+1:K)-1;
hp=2*sin(idx*pi/4) ./ idx / pi; hp(K)=.5; hp=hp/sum(hp);
```

```
Hpf=abs(fft(hp,Lfft));
%===== Drawing the gains
plot(freq,Hif,'-',freq,Hpf,'-b')
%===== Drawing the theoretical frequency response
hold on; plot([0 0.25 .25 .5],[1 1 0 0],':'); hold off
set(gca,'XLim',[0 1/2]); grid
```

Algorithm

To sum up, the window method comprises the following steps:

Steps*:*

1. *Consider the complex gain $H(f)$ we want to implement and the number N of the filter's coefficients.*
2. *The coefficients $h(n)$ are determined by:*

$$h(n) = \int_{-1/2}^{1/2} H(f) e^{2j\pi n f} df \text{ if } N \text{ is odd}$$

$$h(n) = \int_{-1/2}^{1/2} H(f) e^{-j\pi f} e^{2j\pi n f} df \text{ if } N \text{ is even}$$

   and we then calculate N values symmetrically spaced-out around $n = 0$.

3. *If needed, the resulting sequence is multiplied, term-by-term, by a sequence $w(n)$ called a* weighting window*.*

Figure 4.17 shows a comparison of the answers calculated by the `evenodd.m` program.

Type of filters obtained with weighting windows

Up until now, we have designed two types of filters for which the frequency response was 1 or $e^{-j\pi f}$. We also could have considered a complex gain sign(f) or sign$(f)e^{-j\pi f}$. If we assume that we have made the impulse responses $\{h(n)\}$ causal, we end up with four possibilities, four *types*, depending on the characteristics of the impulse responses:

1. type I: N odd and $h(n) = h(N-1-n)$;
2. type II: N even and $h(n) = h(N-1-n)$;
3. type III: N odd and $h(n) = -h(N-1-n)$;

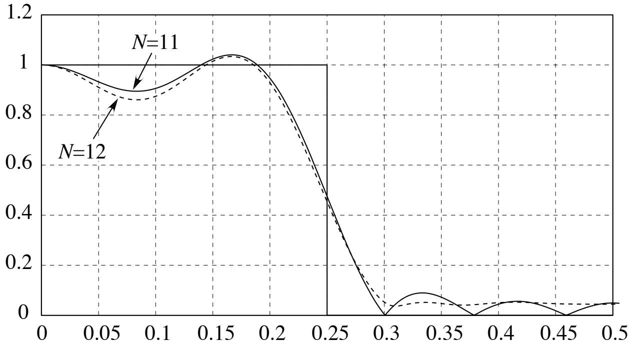


Figure 4.17 – *Half-band filter: frequency response for $N = 11$ and $N = 12$*

4. type IV: N even and $h(n) = -h(N-1-n)$.

We will denote by $P = \lfloor N/2 \rfloor$ the integer part of $N/2$:

1. In the first, by a direct calculation:

$$\begin{aligned} H(f) &= \sum_{n=0}^{P-1} h(n)e^{-2\pi jnf} + h(P)e^{-2\pi jPf} + \sum_{n=P+1}^{N-1} h(n)e^{-2\pi jnf} \\ &= \sum_{n=0}^{P-1} h(n)e^{-2\pi jnf} + h(P)e^{-2\pi jPf} + \sum_{m=0}^{P-1} h(m)e^{-2\pi j(N-1-m)f} \\ &= e^{-\pi j(N-1)f}\left(2\sum_{n=0}^{P-1} h(n)\cos \pi f(N-1-2n) + h(P)\right) \\ &= e^{-\pi j(N-1)f} H_I(f) \end{aligned}$$

$H_I(f)$ is the resulting filter when the coefficients $h(n)$ are chosen symmetrically about $n = 0$ (hence before making the sequence causal). For a low-pass, as we have already seen, the result is:

$$h(n) = \int_{-f_c}^{f_c} Ae^{2\pi jnf}\,df = A\frac{\sin(2\pi n f_c)}{n\pi}$$

2. In the second case:

$$\begin{aligned} H(f) &= \sum_{n=0}^{P-1} h(n)e^{-2\pi jnf} + \sum_{n=P}^{N-1} h(n)e^{-2\pi jnf} \\ &= e^{-\pi j(N-1)f}\left(2\sum_{n=0}^{P-1} h(n)\cos\pi f(N-1-2n)\right) \\ &= e^{-\pi j(N-1)f} H_{II}(f) \end{aligned}$$

For a low-pass, we get:

$$h(n) = \int_{-f_c}^{f_c} Ae^{-j\pi f}e^{2\pi jnf}\,df = A\frac{\sin(2\pi nf_c - \pi f_c)}{n\pi - \pi/2}$$

3. In the third case $h(P) = 0$ and:

$$\begin{aligned} H(f) &= \sum_{n=0}^{P-1} h(n)e^{-2\pi jnf} + h(P)e^{-2\pi jPf} + \sum_{n=P+1}^{N-1} h(n)e^{-2\pi jnf} \\ &= e^{-\pi j(N-1)f}\left(2j\sum_{n=0}^{P-1} h(n)\sin\pi f(N-1-2n)\right) \\ &= e^{-\pi j(N-1)f} H_{III}(f) \end{aligned}$$

For a low-pass (defined by $|H(f)| = 1$), we get:

$$h(n) = \int_{-f_c}^{f_c} jA\ \mathrm{sign}(f)e^{2\pi jnf}\,df = A\frac{\cos(2\pi nf_c) - 1}{n\pi}$$

$H(f)$ shows a "discontinuity" at the origin (gain = 0 for $f = 0$).

4. In the fourth case:

$$\begin{aligned} H(f) &= \sum_{n=0}^{P-1} h(n)e^{-2\pi jnf} + \sum_{n=P}^{N-1} h(n)e^{-2\pi jnf} \\ &= \sum_{n=0}^{P-1} h(n)e^{-2\pi jnf} + \sum_{m=0}^{N-1-P} h(N-1-m)e^{-2\pi j(N-1-m)f} \\ &= e^{-\pi j(N-1)f}\left(2j\sum_{n=0}^{P-1} h(n)\sin\pi f(N-1-2n)\right) \\ &= e^{-\pi j(N-1)f} H_{IV}(f) \end{aligned}$$

For a low-pass, we get:

$$h(n) = \int_{-f_c}^{f_c} jA \text{ sign}(f) e^{-j\pi f} e^{2\pi jnf} df = A\frac{\cos(2\pi n f_c - \pi f_c) - 1}{2n\pi - \pi}$$

$H(f)$ shows a "discontinuity" at the origin as for the previous case.

When designing a low-pass filter, it is actually preferable to choose types I and II. However, if the frequency response has to be asymmetrical, types III and IV can be used.

As an example, type:

```
%===== LOWPASS2.M
clear; nfft=256; freq=[0:nfft-1]'/nfft;
fc=1/8;           % Low-pass filter [-fc,+fc]
Nt=56;
%===== Type I (hn odd) =============
Mt=Nt+1; M=floor(Mt/2);
n=[-M:M]';  % with 2M+1=Mt coefficients
hnI=sin(2*pi*n*fc) ./ n/pi; hnI(M+1)=2*fc;
hnIs=fft(hnI,nfft); hrIs=abs(hnIs);
%===== Type II (hn even) ===============
Mt=Nt; nII=[-Mt/2+1:Mt/2]';
hnII=sin(2*pi*nII*fc-(pi*fc)) ./ (nII*pi-(pi/2));
hnIIs=fft(hnII,nfft); hrIIs=abs(hnIIs);
%===== Type III (hn odd) ============
Mt=Nt+1; nIII=n;
hnIII=(cos(2*pi*nIII*fc)-1) ./ (nIII*pi); hnIII(M+1)=0;
hnIIIs=fft(hnIII,nfft); hrIIIs=abs(hnIIIs);
%===== Type IV (hn even) =============
Mt=Nt; nIV=nII;
hnIV=2*(cos(2*pi*nIV*fc-pi*fc)-1) ./ (2*nIV*pi-pi);
hnIVs=fft(hnIV,nfft); hrIVs=abs(hnIVs);
subplot(211); plot([hnI [hnII;0] hnIII [hnIV;0]]); grid
subplot(212); plot(freq,[hrIs hrIIs hrIIIs hrIVs]);
set(gca,'xlim',[0 .5]); grid;
```

To sum up, if we choose the gain $A(f)$, the impulse response calculations are done by:

1. type I: $h(n) = \int_{-f_c}^{f_c} A(f) e^{2\pi jnf} df$;
2. type II: $h(n) = \int_{-f_c}^{f_c} A(f) e^{-j\pi f} e^{2\pi jnf} df$;
3. type III: $h(n) = \int_{-f_c}^{f_c} jA(f) \text{ sign}(f) e^{2\pi jnf} df$;
4. type IV: $h(n) = \int_{-f_c}^{f_c} jA(f) \text{ sign}(f) e^{-j\pi f} e^{2\pi jnf} df$.

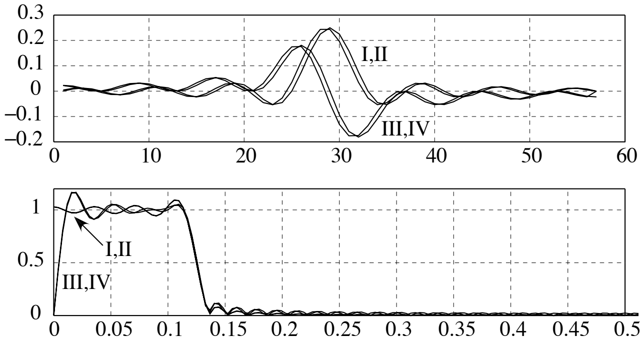


Figure 4.18 – *Impulse responses and gains for the four types of low-pass filters*

Weighting window

Keeping only a finite number of terms of the impulse response $\{h(n)\}$ amounts to multiplying it by an N width rectangle, and therefore to convoluting $H(f)$ with the function:

$$W_r(f) = \sum_{n=0}^{N-1} e^{-2j\pi nf} = \frac{1-e^{-2j\pi Nf}}{1-e^{-2j\pi f}} = \frac{\sin(N\pi f)}{\sin(\pi f)} e^{-j\pi(N-1)f}$$

This weighting function, called the *rectangular weighting window*, causes ripples in the bandpass and the stopband, and widens the transition band, hence the idea of applying other windows so as to modify these properties. Among the most common ones, we can mention the *Bartlett*, *Hamming*, *Hann*, and *Kaiser* windows [74], etc. Let us examine the *Hamming window* defined by:

$$w_H(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right), \text{ for } n \in \{0, \dots, N-1\} \tag{4.37}$$

We plotted in Figure 4.19 the gain of an ideal half-band filter as well as the gains of the filters obtained by the window method, with the rectangular window and the Hamming window respectively.

Notice that reducing the ripples in the *passband* and in the *stopband* causes the *transition band* to widen, in other words the gain decreases more slowly around the frequency $f = 1/4$.

Figure 4.19 is obtained using the program:

```
%===== RIFHAM.M
Lfft=1024; freq=[0:Lfft-1]' / Lfft; N=15; K=(N-1)/2;
%===== Rectangular window
h=sin((1:K)*pi/2) ./ ((1:K))/pi; h=[h(K:-1:1) 1/2 h];
```

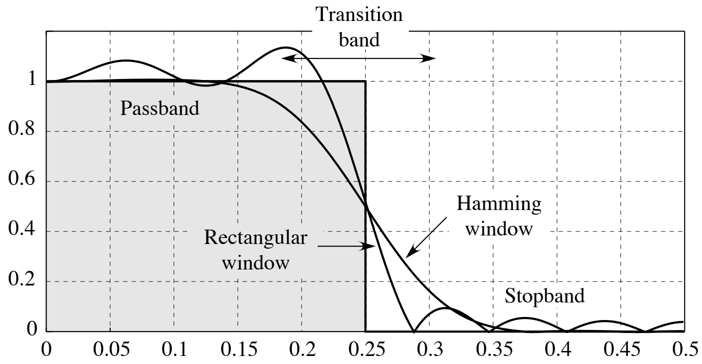


Figure 4.19 – *Gain of a* 15*th order filter for a rectangular window and a Hamming window*

```
h=h/sum(h); Hf=abs(fft(h,Lfft));
%===== Hamming window
ham=.54-.46*cos(2*pi*(0:N-1)/(N-1)); hh=h .* ham;
hh=hh/sum(hh); Hfh=abs(fft(hh,Lfft));
plot(freq,[Hf;Hfh]); axis([0 1/2 0 1.4]); grid
%===== Drawing the theoretical frequency response
hold on; plot([0 0.25 .25 .5],[1 1 0 0],':'); hold off
```

The method gives rise to the following comments:

- the integral we wish to calculate requires the analytical expression of $H(f)$;

- the resulting filter is stable (by definition, since it has no poles);

- because the impulse response is symmetrical, the transfer function has zeros on both sides of the unit circle. Therefore, it is not a minimum phase filter;

- its phase can be linear (piecewise), unlike the IIR;

- the ripples do not have a constant amplitude;

- as we saw with the low-pass filter, the passband and stopband ripples are the same;

- *transition bands* are widened because of the width of the chosen window's main lobe;

- unwanted *ripples* are due to side-lobes.

NOTE: the window method is mainly used because it can be used to provide linear phase filters. This property has to do with the symmetry of the coefficients: it is important for the weighting window to preserve this symmetry. In particular, the two extreme values, for $n = 0$ and $n = N - 1$, are equal (see expression 4.37). Bear in mind that the windows used to weight the signals do not have this symmetry, but are periodic with period N.

Exercise 4.8 (Window method: low-pass filter)
We want to design an ideal low-pass filter, the gain of which is represented in Figure 4.20.

1. Determine the expression of $h(n)$ in the cases where the length is chosen even, and where it is chosen odd.

2. Write the `rif(N,f0)` function, which determines the filter's coefficients based on the length N and the cancelling frequency f_0. Write this function by implementing a Hamming window.

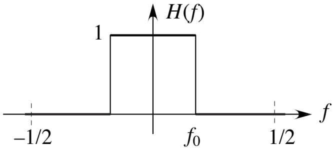


Figure 4.20 – *Ideal low-pass*

Using the `angle` function, use a program to check that the phase is piecewise linear.

Exercise 4.9 (Spectrum reversal encryption)
Starting off with a sound (B band real signal $x(t)$), a typical encryption technique consists of performing the process represented in Figure 4.21.

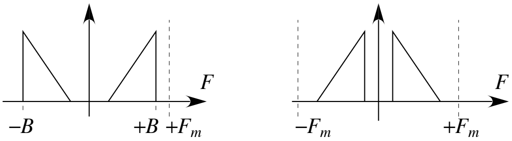


Figure 4.21 – *Sound encryption: on the left, the spectrum of the real signal about to be encrypted; on the right, the encrypted signal's spectrum*

1. Let us assume that $y(t) = 2 \times x(t) \cos(2\pi F_m t)$. Using the spectrum of $y(t)$ shows that, if $F_m > B$, the "encrypted" signal's spectrum can be obtained (see exercise 3.7).

What does the decryption operation consist of?

2. Write a program that decrypts, at the frequency $F_m = 12{,}800$ Hz, a B band signal sampled at F_s.

Exercise 4.10 (Window method: band-pass filter)
Let $h(n)$ be the impulse response of a filter.

1. What is the complex gain of the filter $2h(n)\cos(2\pi n f_0)$? How can a band-pass filter be designed using a low-pass filter.

2. Write a program that uses the `fir` function to design a band-pass filter centered at 0.2 with a width of 0.1.

Exercise 4.11 (Window method: derivative filter)
Let $x(t)$ be a continuous-time signal and let $X(F)$ be its Fourier transform.

1. Show that the derivative of the Fourier transform of $x(t)$ has the expression $2j\pi F X(F)$. Using a procedure similar to the one found on page 133, determine the complex gain of a discrete-time derivative linear filter.

2. Using this result and the window method, find the coefficients of the FIR filter that approximates a derivative filter for an odd number of coefficients.

3. The resulting filter is not causal. Give a causal solution to the problem. What is the consequence on the operation performed?

4. Write the derivative filter as a MATLAB® function and test it on different types of signals, in particular on a signal of the type $\sin(2\pi f_0 n)$.

4.7.3 IIR filter design

A rather common method of recursive filter design is based on the continuous-time/discrete-time change. Starting off with a “continuous-time” filter, the characteristics of which are known, all that needs to be done is to “discretize” that filter. This isn’t the only way to proceed of course. Other design methods can be found in more specialized books, in particular methods that do not require changing over to continuous-time (notably the Remez method).

A few characteristics of the analog filters used

1. The *Butterworth filters* have a continuous-time transfer function $H(s)$ (Laplace transform of their impulse response) resulting from a gain function that can be expressed:

$$|H_f(f)|^2 = \frac{1}{1+(2\pi f)^{2n}} \Rightarrow H(s) = \frac{1}{\prod_{k=1}^{n}(s-s_k)}$$

where $s_k = j\pi\left(\frac{2k-1}{2n}+\frac{1}{2}\right)$. The situation changes depending on whether n is odd or even:

$$H_{\text{even}} = \prod_{k=1}^{n/2} \frac{1}{s^2+2s\cos((2k-1)\pi/2n)+1}$$

$$\text{and } H_{\text{odd}} = \frac{1}{s+1}\prod_{k=1}^{(n-1)/2} \frac{1}{s^2+2s\cos(k\pi/n)+1}$$

Butterworth filters show no ripples in their passband and stopband.

2. Type 1 Chebyshev filters have the gain $|H_f(f)|^2 = (1+\varepsilon^2 T_n^2(f))^{-1}$ with:

$$T_n(x) = \begin{cases} \cos(n\arccos(x)) & \text{for} \quad |x|<1 \\ \cosh(n\operatorname{arg}\cosh(x) & \text{for} \quad |x|>1 \end{cases}$$

And hence for $k \in \{1,\cdots,n\}$, $p_k = -\sinh(\alpha)\sin((2k-1)\pi/2n) + j\cosh(\alpha)\cos((2k-1)\pi/2n)$ with $\alpha = \operatorname{arg}\sinh(\varepsilon^{-1})/n$.

3. Type 2 Chebyshev filters have the following gain:

$$|H_f(f)|^2 = \frac{1}{1+\varepsilon^2 T_n^2(f_a)/T_n^2(f)}$$

where f_a is the frequency where the stopband begins. And therefore:

$$p_k = \frac{2\pi f_a A_k}{A_k^2+B_k^2} - j\frac{2\pi f_a B_k}{A_k^2+B_k^2} \text{ for } k \in \{1,\cdots,n\}$$
$$\text{with}\begin{cases} A_k = -\sinh(\alpha)\sin((2k-1)\pi/2n) \\ B_k = \cosh(\alpha)\cos((2k-1)\pi/2n) \end{cases}$$

$\alpha = \operatorname{arg}\cosh(A_a^{-1})/n$, where$A_a$ is the imposed passband amplitude. The zeros are placed on the imaginary axis:

$$z_k = \frac{2j\pi f_a}{\cos((2k-1)\pi/2n)}$$

Chebyshev filters show:

– either a passband ripple but none in the stopband (type 1 filters);

– either a stopband ripple but none in the passband (type 2 filters).

The passband ripple is equal to $1/\sqrt{1+\varepsilon^2}$. The Chebyshev filter have better attenuation characteristics than the Butterworth filters. The bandpass ripple, the attenuation and the integer n all need to be known in order to calculate these filters.

4. *Cauer filters* or *elliptic filters* are optimal in terms of the transition band, and have the following gain:

$$|G(f)|^2 = \frac{1}{1+\varepsilon^2 R_n^2(f, L)}$$

where R_n is a rational Chebyshev approximation, and where L characterizes the attenuation. The Cauer filters show ripples in the bandpass and in the stopband. The band pass ripple is equal to $1/\sqrt{1+\varepsilon^2}$. The bandpass ripple and the minimum attenuation in the stopband are needed to be able to calculate these filters,which are defined by arrays.

Using the bilinear transform

Using the bilinear transform is justified by the calculation of an integral with the trapezoid method. We will write $x_n = x(nT)$ to denote the function's values at points nT. If s_n is the value of the integral from 0 to nT, s_n obeys the recursive equation:

$$s_n = s_{n-1} + T\frac{x_n + x_{n-1}}{2} \quad \Rightarrow \quad B(z) = \frac{S(z)}{X(z)} = \frac{T}{2}\frac{1+z^{-1}}{1-z^{-1}}$$

$B(z)$ is an approximation of the integral operator, which is expressed $1/s$ in the Laplace transform. The bilinear transform method consists of replacing s with $1/B(z)$ in the expression of the continuous-time filter's transfer function. [27, 8]. If the sampling is done fast enough, the frequency distortion caused by this transformation is negligible. When this is not the case, there are methods for compensating this distortion (to a certain extent).

Exercise 4.12 (Butterworth filter)
We wish to perform an IIR filter design based on the Butterworth filter.

1. Write a program designed to calculate the coefficients of the Butterworth filter's denominator, for a given value of n.

2. Write the program that provides the frequency response of the Butterworth filter for a few orders (for example, for n from 2 to 6). The gain, in decibels, will be chosen as the y-coordinate, and $\log_{10}(\omega)$ as the x-coordinate, where ω is the angular frequency in rad/s (for a given value of ω, the value of a polynomial is the scalar product of its coefficients with the vector $[1 \; j\omega \cdots (j\omega)^N]$).

3. Using the bilinear transform to obtain the digital filter:
   (a) Consider the polynomial $G(x) = a_0 + a_1 x + \cdots + a_N x^N$. Write its development as a recursive relation based on the Horner polynomial representation[5].
   (b) The variable change $x = B(z)/A(z)$ is made, where $A(z)$ and $B(z)$ are polynomials. Write the previous relation for the numerator and denominator polynomials.
   (c) Write the program that performs the bilinear transform of a polynomial.
4. Compare the obtained discrete and continuous spectra.

Using the DFT

A method that would seem reasonable would be, for filter design, to start with values of the DTFT $H(f)$ (actually the values of the DFT), and to try to find the original $\{h(n)\}$ using the IDFT. As we are going to see in exercise 4.13, it actually is not such a good idea.

Exercise 4.13 (Temporal aliasing and the use of the DFT)
Consider $H(f)$, the complex gain of a filter with the impulse response $\{h(n)\}$. Give the expression of the filter's impulse response $\{\tilde{h}(n)\}$, calculated based on the DFT $H(k/N)$ as a function of the $h(n)$. Compare the results for the window method and this method, on a low-pass filter. Compare the resulting gains.

4.8 Oversampling and undersampling

The oversampling and undersampling operations play an important role in digital signal processing.

Oversampling by an integer factor M consists of performing an interpolation on the sequence $x(n)$ by calculating $M - 1$ intermediate values between two consecutive points.

Undersampling by an integer factor M consists of calculating, based on a sequence sampled at the frequency F_s, the values of that same sequence as if it had been sampled at F_s/M. Undersampling does not mean simply taking one out of every M samples of the original sequence.

A typical application of oversampling and undersampling is the frequency change. In order to go from 42 kHz to 48 kHz, for example, you can start by oversampling by a factor of 8, and then undersample by a factor of 7. These

[5] A polynomial $x^n + a_1 x^{n-1} + \cdots + a_n$ can be developed by writing $x(\cdots(x(x + a_1) + a_2)\cdots) + a_n$. This is known as the Horner polynomial representation or the Horner scheme.

operations are also helpful for what is called "multifrequency" processing, which is used in particular for bank filter techniques, presented in paragraph 5.2.

4.8.1 Oversampling

We are going to start with an example. Let $x(n)$ be a sequence with $X_z(z)$ as its z-transform. Consider the sequence $y(n) = x(n/4)$ for $n = 0 \bmod 4$ and $y(n) = 0$ if $n \neq 0 \bmod 4$. This operation, called the *expansion* operation, inserts three 0 in between the terms of the sequence $x(n)$. Notice that the sequence $y(n)$ contains 4 times more samples than the sequence $x(n)$, and theoretically, should be interpolated at 4 times the sampling frequency F_s associated to $x(n)$.

We will now determine the expression of the z-transform of the sequence $y(n)$ as a function of $x(n)$. With obvious notations, we have:

$$Y_z(z) = \sum_k x(n)z^{-4n} = X_z(z^4)$$

With $z = e^{2j\pi f}$ the DTFT can be expressed as $Y(f) = X(4f)$. The interval $(-1/2, 1/2)$ contains the spectrum of $x(n)$ replicated four times. We say that there are *images* in the spectrum (see Figure 4.22).

This means that if you consider the samples $y(n)$ corresponding to a continuous-time signal sampled at the frequency $F'_s = 4F_s$, the spectrum is made up of these images in the $(-2F_s, +2F_s)$ band (see page 69). Inserting zeros where real values should be has added high frequency components corresponding to the brutal transitions introduced in the temporal sequence.

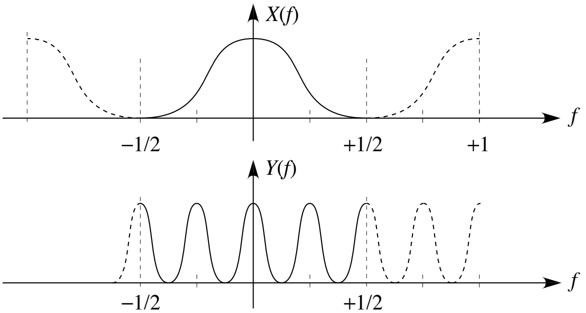


Figure 4.22 – *Effects of oversampling*

If we have a sequence $y(n)$ and wish to reconstruct the intermediate samples without causing the distortion due to the images, we simply need, after the expansion operation, to perform a gain $F'_s/F_s = 4$ filtering in the $(-1/8, 1/8)$ band, in order to suppress the high frequencies found in the $(-1/2, -1/8)$ and $(+1/8, +1/2)$ bands. The resulting signal, after being reconstructed at the frequency $4F_s$, is located in the $(-4F_s/8, +4F_s/8) = (-F_s/2, +F_s/2)$ band, with 4 times the number of samples.

All these results can easily be generalized. If we insert $(M-1)$ zeros in between the elements of the sequence $x(n)$, the resulting sequence's z-transform has the expression:

$$Y_z(z) = X_z(z^M) \tag{4.38}$$

In order to properly oversample, this expansion has to be followed by a low-pass filter in the $(-1/2M, 1/2M)$ band, corresponding, after reconstruction at the frequency $F_s' = MF_s$, to the original $(-F_s/2, F_s/2)$ band with M times the number of samples. What should be remembered of all this is written below:

In order to oversample a signal $x(n)$ by a factor M, one method consists of inserting $(M-1)$ zeros in between the values of the signal, and then to perform an M gain filtering in the $(-1/2M, +1/2M)$ band.

Exercise 4.14 (Interpolation)

1. Write a function that interpolates by a factor of M.
2. Apply this function to the `x=rand(1,40);` sequence.

Example 4.9 (Expansion and frequency translation) Starting off with a speech signal the spectrum of which is placed, for the positive frequencies, in the (0 Hz - 4,000 Hz) band, we are going to listen to the signal resulting from the following operations:

1. The frequency scale is expanded by a factor of 5/4. Mathematically speaking, this means that if $S_x(F)$ refers to the original signal's spectrum, the modified signal's spectrum is $S_y(F) = S_x(5F/4)$. Therefore, the spectrum can be found, for the positive frequencies, in the (0 Hz - 5,000 Hz) band. How is this achieved?
2. The signal's spectrum is shifted by $F_0 = 1$ kHz toward the positive frequencies. This means that if $S_x^+(F)$ refers to the part of the signal's spectrum found in the positive frequencies of the original signal, then the part belonging to the positive frequencies of the modified signal's spectrum is $S_y^+(F) = S_x^+(F-F_0)$. Hence the spectrum is now, for the positive frequencies, in the (1, 4) kHz band. How is this achieved?

Write a program using MATLAB® that performs these two operations. Listen to the obtained signals and compare the results. You can also compare the spectra with the **smperio.m** function discussed in Chapter 9, example 9.1.

HINT:

1. According to the time scale expansion/compression property (see Chapter 1), all that is needed to expand the spectrum is to take the signal sampled at the frequency $F_s = 8{,}000$ Hz, and listen to it at the reconstruction frequency $\frac{5}{4}F_s$ Hz. Type:

   ```
   soundsc(x, 10000);
   ```

   In order to compare with the signal obtained in the next question, you can also construct the signal interpolated by a factor of 2 corresponding to the sampling frequency $F_s' = 2F_s = 16{,}000$ Hz and listen to the result at the frequency $\frac{5}{4}F_s' = 20{,}000$.

2. The original signal, sampled at $F_s = 8{,}000$ Hz, is in the $(-4, +4)$ kHz band. Because of the frequency translation, the desired signal is in the $(-5, +5)$ kHz band. Hence interpolation must first be performed in order to have a sampling frequency at least equal to 10,000 Hz. To make the interpolation operations simpler, we will set $F_s' = 16{,}000$ Hz, by using the **interpM** function with an interpolation factor of 2.

   The spectrum has then to be shifted by 1 kHz. This can be done by multiplying the signal by the function $e^{2j\pi 1000t}$ sampled at the frequency $F_s' = 16{,}000$ Hz.

   The $(+1, +5)$ kHz band then has to be filtered. To this purpose, a low-pass filter is implemented in the $(-2, +2)$ kHz band and shifted in frequency by 3 kHz. This result is achieved using the **rif** function in the $(-b, b)$ band where $b = \frac{2000}{16000}$ followed by a multiplication of the filter coefficients by $e^{2j\pi n3000/16000}$.

■

Type the program:

```
%===== FRQSHIFT.M
close all; clear all
Fs=8000; load phrase.mat; % or [sn,Fs]=wavread('phrase.wav');
Fep=2*Fs; xi2=interM(sn,2,100);
F0=1000; xi2trans=xi2.*exp(2*j*pi*(0:length(xi2)-1)'*F0/Fep);
%===== Low-pass filter
Lh=201; h=rif(Lh,Fs/4/Fep);
%===== Band-pass filter centered on Fc
Fc=Fs/4+F0; htrans=h .* exp(2*j*pi*(0:Lh-1)*Fc/Fep);
```

```
xi2transfiltre=filter(htrans,1,xi2trans);
xtrans=real(xi2transfiltre);
%===== Listening (sound or soundsc depending on the version)
soundsc(sn,Fs); disp('Press a key'); pause
soundsc(sn,5*Fs/4); disp('Press a key'); pause
soundsc(xtrans,Fep);
```

Digital-to-analog conversion

In paragraph 2.1.2, we saw that distortions appear during the signal's reconstruction simply by using a ZOH. In the case of audio frequency applications (speech, music, etc.), a simple way of avoiding this is to place before the ZOH an oversampler with a high enough factor M.

This is because oversampling spreads further apart the periodized components of the signal's spectrum. Furthermore, concerning the ZOH, working M times faster "widens" the sine cardinal lobes.

Figure 4.23 shows what the spectrum looks like for the output signal of the ZOH, for $M = 5$ and for a sampling frequency of $F_s = 1/T = 8$ kHz. The result should be compared to the one in Figure 2.9 of paragraph 2.1.2.

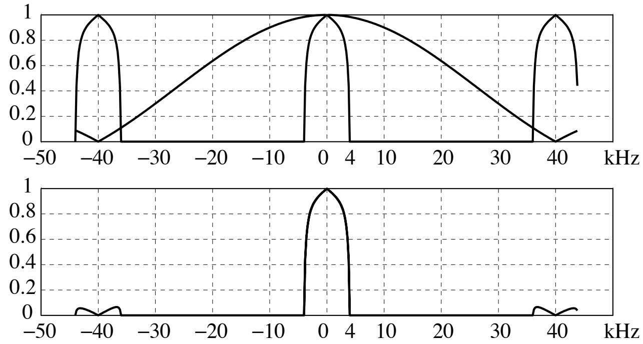


Figure 4.23 – *Output spectrum of a ZOH preceded by an oversampling with a factor of* $M = 5$

The part found in the $(-F_s/2, F_s/2)$ undergoes a slight distortion, since $\text{sinc}(fT/M)$ stays close to 1. Beyond $F_s/2$, the first term due to the periodization of the spectrum can be found around 40 kHz, outside of the audible band. This method is often used by the boards installed in our computers to avoid having to use an analog high-quality low-pass filter: the signal is oversampled by a factor M so as to have $M \times F_s$ greater than 40 kHz (typically, $M \times F_s \approx 48$ kHz), and the obtained values are maintained constant, at the same processing rate.

4.8.2 Undersampling

Let $\{x(n)\}$ be a sequence with $X(z)$ as its z-transform. Consider the sequence $\{y(n) = x(4n)\}$ obtained by keeping only one out of every 4 samples of the sequence $\{x(n)\}$. The operation that takes us from $\{x(n)\}$ to $\{y(n)\}$ is called a *factor $M = 4$ decimation.*

By using the identity 2.33, the expression of the z-transform of $\{y(n)\}$ is:

$$\begin{aligned} Y_z(z) &= \sum_{n=-\infty}^{+\infty} x(4n)z^{-n} = \sum_{p=-\infty}^{+\infty} x(p)\left(\frac{1}{4}\sum_{k=0}^{3} e^{2j\pi kp/4}\right) z^{-p/4} \\ &= \frac{1}{4}\sum_{k=0}^{3}\left(\sum_{p=-\infty}^{+\infty} x(p)\left(z^{1/4}e^{-2j\pi k/4}\right)^{-p}\right) = \frac{1}{4}\sum_{k=0}^{3} X_z\left(z^{1/4}e^{-2j\pi k/4}\right) \end{aligned}$$

Notice that you must not write $p = 4n$ and then $p \in \mathbb{Z}$. The resulting *inexact expression* would be $Y_z(z) = X_z(z^{1/4})$. By using the DTFT, that is to say by choosing $z = e^{2j\pi f}$ and by recalling the notation $X(f) = X_z(e^{2j\pi f})$, we get, for $M = 4$:

$$Y(f) = \frac{1}{4}\sum_{k=0}^{3} X\left(\frac{f-k}{4}\right) \tag{4.39}$$

This expression shows that inside the interval $(-1/2, 1/2)$, $Y(f)$ is the algebraic sum of the four contributions, shifted by $1/4$. To obtain the signals of the continuous-time signals (frequencies expressed in Hz), f has to be multiplied by the sampling frequency as we have already said. The $y(n)$ are the signal samples taken at the frequency $F_s/4$, hence the expression of $Y(f)$ shows a *spectrum aliasing* effect. Notice, by the way, the difference with the expansion operation that creates "images" in the signal's spectrum.

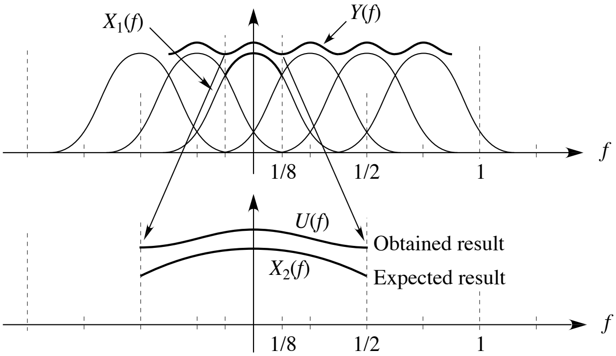


Figure 4.24 – *Effects of undersampling*

However, according to the sampling theorem, the undersampled signal's spectrum is the one referred to as $X_2(f)$ in Figure 4.24. Therefore, to *undersample* the signal $x(n)$, a gain 1 filtering (see page 69 with the ratio $F'_s/F_s = 1/M$ already included in 4.39) has to be performed in the $(-1/8, +1/8)$ band *before* the decimation operation, so as to avoid spectrum aliasing.

These results can be generalized. If $y(n)$ refers to the sequence obtained by taking one out of every M values of the sequence $x(n)$, the expression of its z-transform is:

$$Y_z(z) \quad = \quad \frac{1}{M}\sum_{k=0}^{M-1} X_z\left(z^{1/M}e^{-2j\pi k/M}\right) \tag{4.40}$$

In order to undersample *a signal $x(n)$ by a factor M, one possible method is to perform a gain* 1 *filtering in the* $(-1/(2M), +1/(2M))$ *band, followed by a decimation operation of* 1 *out of every M values.*

Exercise 4.15 (Undersampling)

1. Write a function for undersampling by a factor M.
2. Record a speech signal at 8,000 Hz.
   - Create a new signal by taking one out of every 2 samples without any particular processing. Listen to the result.
   - Perform a "proper" undersampling by using the previous function with $M = 2$. Listen to the result.

Figure 4.25 sums up the M factor oversampling and undersampling operations. It should be noted that *all* of these operations, including the filtering, are performed in discrete-time, and on no occasion did we change to the continuous-time signal!

Exercise 4.16 (Paralleled undersampling and oversampling)
The filtering operation necessary to the undersampling can be performed M times faster by M parallel filters. The output at the time nM has the expression:

$$\begin{aligned} y(nM) &= \sum_{k=-\infty}^{+\infty} h(k)x(nM-k) \\ &= \sum_{r=0}^{M-1}\sum_{m=-\infty}^{+\infty} h(mM+r)x((n-m)M-r) \end{aligned}$$

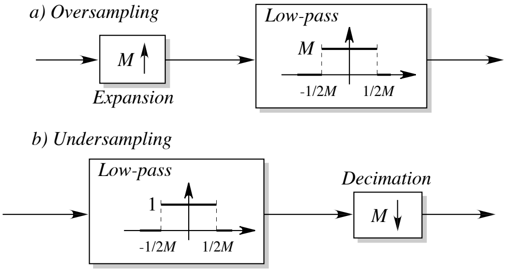


Figure 4.25 – *Oversampling and undersampling operations*

$y(nM)$ appears as the sum of M filterings with the impulse responses $\{h_r(m) = h(mM + r)\}_{m\in\mathbb{Z}}$. The filter input is the sequence $\dots$, $x(r - M)$, $x(r)$, $x(r + M)$, $x(r + 2M)$,...obtained from $x(n)$ by a delay of r followed by a decimation. Notice that h_r is the r-th M-polyphase component of h.

1. Give the processing architecture.
2. Write a paralleled undersampler simulation program.
3. Write a paralleled oversampler simulation program.

This page intentionally left blank

Chapter 5

Filter Implementation

Mathematically speaking, using the filter with the transfer function $H(z)$ for filtering the sequence $x(n)$ leads to a perfectly determined result. However, depending on the practical implementation of the filter, the results can vary in terms of precision, speed, etc. This chapter deals with the technical aspects of filtering. If you restrict yourself to a "simulation" approach, as we have up until now, the `filter` function is everything you will ever need. However, if this filtering operation has to be implemented, its effectiveness requires some additional knowledge that will be detailed in this chapter.

5.1 Filter implementation

5.1.1 Examples of filter structures

In this paragraph, we will study the implementation of the filtering function, in other words its programming. Figure 5.1 shows a particular implementation called the *canonical direct form* of a general recursive filter with the transfer function:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_p z^{-p}}{1 + a_1 z^{-1} + \cdots + a_p z^{-p}}$$

Choosing the same degree for both the numerator and the denominator does not restrict us in any way; you need only consider that some of the coefficients can be equal to zero.

This "implementation" first performs the calculation of:

$$t(n) = i(n) - a_1 t(n-1) - \cdots - a_p t(n-p)$$

then the calculation of:

$$o(n) = b_0 t(n) + b_1 t(n-1) + \cdots + b_p t(n-p)$$

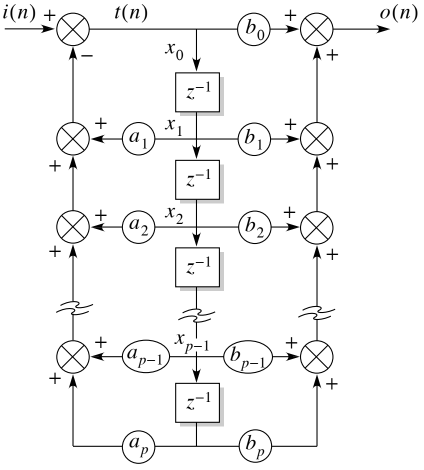


Figure 5.1 – *Processing architecture*

where $\{i(n)\}$ and $\{o(n)\}$ are the input and output sequences respectively.

For this algorithm, the vector $\mathbf{x}(n) \overset{\Delta}{=} \begin{bmatrix} x_0(n) & x_1(n) & \dots & x_{p-1}(n) \end{bmatrix}^T$ $(p \times 1)$ is called the *filter state*:

$$\mathbf{x}(n) = \begin{bmatrix} t(n) & t(n-1) & \dots & t(n-p+1) \end{bmatrix}^T$$

Its components, referred to as *state variables*, are the input values of the "delay" cells denoted z^{-1} in Figure 5.1. By introducing the vectors:

$$\mathbf{a} = [a_1 \quad a_2 \quad \dots \quad a_p]^T \text{ and } \mathbf{b} = [b_1 \quad b_2 \quad \dots \quad b_p]^T$$

we get the following expression for the algorithm:

$$\left\{ \begin{array}{l} t(n) = i(n) - \mathbf{a}^T\mathbf{x}(n-1) \\ o(n) = b_0 t(n) + \mathbf{b}^T\mathbf{x}(n-1) \\ \mathbf{x}(n) = \begin{bmatrix} t(n) & x_0(n-1) & x_1(n-1) & \cdots & x_{p-2}(n-1) \end{bmatrix} \end{array} \right. \tag{5.1}$$

The following `filter` function implements this algorithm:

```
function [ys,xs] = filtrer(num,den,xe,xi)
%%========================================%
%% Filter (direct canonical structure)    %
%% SYNOPSIS: [ys,xs]=FILTRER(num,den,xe,xi) %
%%        num = [b0 b1 ... bP]            %
%%        den = [1 a1 a2 ... aP]          %
%%        xe  = input sequence            %
%%        xi  = initial state             %
%%        ys  = output sequence           %
%%        xs  = final state                    %
%%===========================================%
lden=length(den); lnum=length(num);
if (lden<lnum), den(lnum)=0; lden=lnum; end
if (lnum<lden), num(lden)=0; end
ld=lden-1; N=length(xe);
av=zeros(1,ld); bv=av;
av(:)=den(2:lden); bv(:)=num(2:lden);
if (nargin==3), zzi=zeros(ld,1); end
if (nargin==4),
    if length(xi)<ld, xi(ld)=0; end
    zzi=zeros(ld,1); zzi(:)=xi;
end
b0=num(1); xs = zzi; ys=zeros(ld,1);
for ii=1:N,
    x0n=xe(ii) - av * xs;
    ys(ii)=b0 * x0n + bv * xs;
    xs=[x0n ; xs(1:ld-1)];      % New state
end
return
```

Determining the initial state leading to a given input-output sequence is another problem altogether. Using the recursive equations 5.1 that lead to $t(n)$ and $o(n)$, we can also write:

$$\begin{bmatrix} o(p-1) \\ i(p-1) \\ \vdots \\ o(0) \\ i(0) \end{bmatrix} = \begin{bmatrix} b_0 & b_1 & \cdots & b_p & 0 & \cdots & \cdots & 0 \\ 1 & a_1 & \cdots & a_p & 0 & \cdots & \cdots & 0 \\ 0 & b_0 & \cdots & b_{p-1} & b_p & 0 & \cdots & 0 \\ 0 & 1 & \cdots & & & & & \\ \vdots & & & & & & & \\ 0 & \cdots & \cdots & 0 & b_0 & b_1 & \cdots & b_p \\ 0 & \cdots & \cdots & 0 & 1 & a_1 & \cdots & a_p \end{bmatrix} \begin{bmatrix} t(p-1) \\ \vdots \\ t(-1) \\ \vdots \\ t(-p) \end{bmatrix} = \mathcal{O}\mathbf{T}$$

This expression shows that the initial state $\mathbf{T} = [t(-1) \ \ldots \ t(-p)]^T$ can be reconstructed so long as the matrix $\mathcal{O}$ is invertible. The system theory demonstrates that the possibility of reconstruction is related to the concept of *observability*. A great number of observability criteria exist, based on the *state representations associated with a system* [51].

The **filtric** function detailed hereafter carries out the reconstruction of the state associated to the processing architecture implemented by **filter**:

```
function zi=filtric(num,den,xi,yo)
%%===========================================%
%% Initial state reconstruction for a direct %
%% canonical structure                       %
%% SYNOPSIS: zi=FILTRIC(num,den,xi,yo)       %
%%          num = [b0 b1 ... bP]             %
%%          den = [1 a1 a2 ... aP]           %
%%        xi  = input sequence            %
%%        yo  = output sequence           %
%%        zi  = reconstructed initial state %
%%========================================%
lden=length(den); lnum=length(num);
if (lden<lnum), den(lnum)=0; lden=lnum; end
if (lnum<lden), num(lden)=0; end
ld=lden-1;
numv=zeros(lden,1); denv=numv;
numv(:)=num; denv(:)=den;
lx=length(xi); ly=length(yo);
if lx<ld, xi(ld)=0; end
if ly<ld, yo(ld)=0; end
ysv=zeros(1,ld); xev=zeros(1,ld);
ysv(:)=yo(ld:-1:1); xev(:)=xi(ld:-1:1);
x=[ysv;xev];  vec=zeros(2*ld,1); vec(:)=x;
v0=[numv; zeros(ld-1,1); denv; zeros(ld,1)];
A=[]; for ii=1:ld, A=[A v0]; end
A=A(1:4*ld*ld);
Ax=zeros(2*ld,2*ld); Ax(:)=A; Ax=Ax';
zzi=inv(Ax) * vec; zi=zzi(ld+1:2*ld);
return
```

The state reconstruction function is inseparably related to the filtering function, which is itself based on a particular processing architecture.

However, as the following example shows, the reconstruction function is rarely used. The state vector **xs**, final state of the first block's processing, is transmitted as the initial state of the second block. The result **yp** is identical to the one obtained for the filtering of the entire block **etot**.

```
%===== FIL2BLOCKS.M
inp1=randn(100,1); inp2=randn(100,1); etot=[inp1;inp2];
b=[1 .3]; a=[1 -.8 .9];
%===== Global filtering  (null initial state)
y=filtrer(b,a,etot);
%===== Filtering the 2 blocks
[y1 xs]=filtrer(b,a,inp1);  % Null initial state
y2=filtrer(b,a,inp2,xs);    % Initial state xs
yp=[y1;y2];
%===== Drawing for the transition between 2 blocks
[y(90:110) yp(90:110)]
```

MATLAB®'s filtering function, **filter**, uses the *Transpose-Form IIR* structure [69], different from the previous one, represented in Figure 5.2. As in our example, `filter` transmits the state vector. It is however impossible to obtain it using the reconstruction function `filtic.m`, available as part of the *Signal Toolbox*. Exercise 5.1 is a study of this structure.

Exercise 5.1 (Filter architecture)
Consider the Transpose-Form IIR structure (Figure 5.2) of a rational filter.

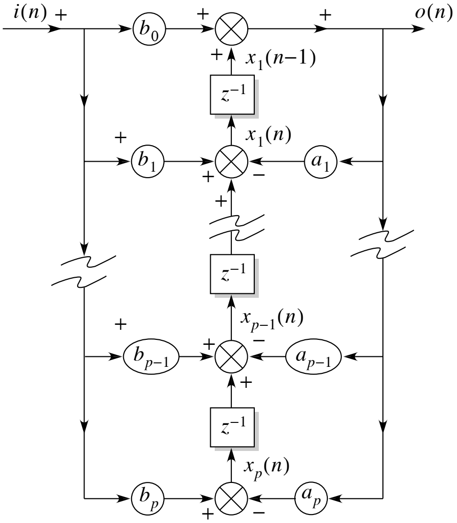


Figure 5.2 – *Transpose-Form IIR structure*

1. Determine the filter's transfer function.

2. By defining the state $\mathbf{x} = [x_1(n) \ \ldots \ x_p(n)]$ at the time n, determine the *state representation* and express it as follows:

$$\begin{cases} \mathbf{x}(n) = \mathbf{A}\mathbf{x}(n-1) + \mathbf{b}i(n) \\ o(n) = \mathbf{c}^T\mathbf{x}(n-1) + di(n) \end{cases}$$

   Use this to find the filtering program. It might be useful to notice that the matrix $\mathbf{A}$ is the transpose of the *companion* matrix (`compan` function) associated with the denominator polynomial $[1 \ a_1 \ a_2 \ \ldots \ a_p]$.

3. Find the associated reconstruction function using only the filtering function. In order to do this, express $x_k(0)$ as the sum of an input filtering and an output filtering.

5.1.2 Distributing the calculation load in an FIR filter

We wish to distribute the calculation load for an FIR filtering algorithm among several processors. Only two methods will be presented. The first one consists of distributing the number of multiplication/accumulation operations (MAC operations) among M branches without changing the processing rate. The second one consists of organizing the calculation in different units, so as to reduce this speed, at the cost of a certain delay.

Paralleled calculations

Consider the filtering equation $y(n) = \sum_{k=-\infty}^{+\infty} h(k)x(n-k)$. For a given M, we define $k = mM + r$ where $r \in \{0, \ldots, M-1\}$. We get:

$$y(n) = \sum_{r=0}^{M-1} \sum_{m=-\infty}^{+\infty} h(mM+r)x(n-mM-r)$$

This expression shows $y(n)$ as the sum of M terms:

$$\begin{array}{r|l} r=0 & \sum_{m=-\infty}^{+\infty} h(mM)x(n-mM) \\ r=1 & \sum_{m=-\infty}^{+\infty} h(mM+1)x(n-mM-1) \\ \vdots & \\ r=M-1 & \sum_{m=-\infty}^{+\infty} h(mM+M-1)x(n-mM-M+1) \end{array}$$

The first term is the filtering of a sequence $\ldots, x(n-M), x(n), x(n+M), \ldots$ by the filter with the impulse response $h(0), h(M), \ldots$ The next terms correspond to translated sequences filtered by the filters $h_r(m) = \{h(r), \ldots, h(r+mM), \ldots\}$. The filter $h_r(m)$ is called the r-th *M-polyphase* component of $h(n)$.

Figure 5.3 illustrates this processing method.

Exercise 5.2 (Parallel implementation of the FIR filtering)
Write a program designed to simulate the process described by Figure 5.3. Choose $M = 4$ and a low-pass, $(-0.3, +0.3)$ band FIR filter with 25 coefficients. The result will be compared to the one obtained through direct filtering.

This method for paralleling does not reduce the processing speed in intermediate filters. Only the number of multiplications per filter is reduced.

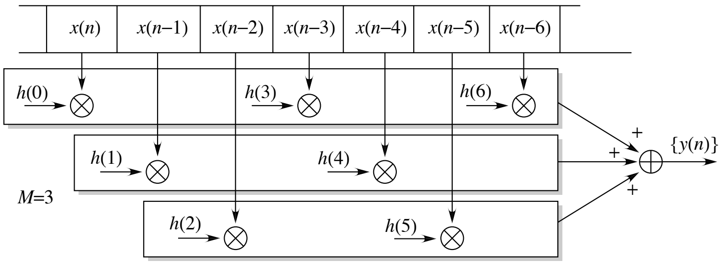


Figure 5.3 – *A representation of the paralleled process*

5.1.3 FIR block filtering

Let us again consider the FIR filtering equation:

$$y(k) = h(0)x(k) + h(1)x(k-1) + \cdots + h(P)x(k-P)$$

Let:

$$\mathbf{y} \;=\; \begin{bmatrix} y(nN) \\ y(nN-1) \\ \vdots \\ y(nN-(N-1)) \end{bmatrix} = \mathbf{X} \times \begin{bmatrix} h(0) \\ h(1) \\ \vdots \\ h(P) \end{bmatrix}$$

with:

$$\mathbf{X} = \begin{bmatrix} x(nN) & x(nN-1) & \cdots & x(nN-P) \\ x(nN-1) & x(nN-2) & \cdots & x(nN-1-P) \\ \vdots & & & \\ x(nN-(N-1)) & x(nN-N) & \cdots & x(nN-(N-1)-P) \end{bmatrix}$$

By organizing the inputs modulo M, we get:

$$\mathbf{y} = \left[\begin{matrix} [x(nN) & x(nN-M) & \cdots &] \\ [x(nN-1) & x(nN-1-M) & \cdots &] \\ \vdots & & & \\ [x(nN-(N-1)) & x(nN-(N-1)-M) & \cdots &] \end{matrix} \right. \quad \cdots$$

$$\cdots \quad \left. \begin{matrix} [x(nN-1) & x(nN-1-M) & \cdots &] & \cdots \\ [x(nN-2) & x(nN-2-M) & \cdots &] & \cdots \\ \vdots & & & & \\ [x(nN-N) & x(nN-N-M) & \cdots &] & \cdots \end{matrix} \right] \times \begin{bmatrix} \begin{bmatrix} h(0) \\ h(M) \\ \vdots \end{bmatrix} \\ \begin{bmatrix} h(1) \\ h(M+1) \\ \vdots \end{bmatrix} \\ \vdots \end{bmatrix}$$

By restricting ourselves to the case $M = N = 2$, the previous expression can be written:

$$\begin{bmatrix} y(2n) \\ y(2n-1) \end{bmatrix} =$$

$$\begin{bmatrix} [x(2n) & x(2n-2) & \cdots &] & [x(2n-1) & x(2n-3) & \cdots &] \\ [x(2n-1) & x(2n-3) & \cdots &] & [x(2n-2) & x(2n-4) & \cdots &] \end{bmatrix} \begin{bmatrix} \begin{bmatrix} h(0) \\ h(2) \\ \vdots \end{bmatrix} \\ \begin{bmatrix} h(1) \\ h(3) \\ \vdots \end{bmatrix} \end{bmatrix}$$

If we assume $\mathbf{x}_0(n) = [x(2n), x(2n-2), \ldots]^T$ and $\mathbf{x}_1(n) = [x(2n-1), x(2n-3), \ldots]^T$, we can also write:

$$\begin{bmatrix} y(2n) \\ y(2n-1) \end{bmatrix} = \begin{bmatrix} \mathbf{x}_0(n) & \mathbf{x}_1(n) \\ \mathbf{x}_1(n) & \mathbf{x}_0(n-1) \end{bmatrix} \times \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}$$

where $\mathbf{H}_0 = [h(0), h(2), \ldots]^T$ and $\mathbf{H}_1 = [h(1), h(3), \ldots]^T$. If we develop $y(2n)$ and $y(2n-1)$, we get:

$$\begin{aligned} y(2n) &= \mathbf{x}_0(n)\mathbf{H}_0 + \mathbf{x}_1(n)\mathbf{H}_1 \\ &= \mathbf{x}_1(n)(\mathbf{H}_0 + \mathbf{H}_1) + (\mathbf{x}_0(n) - \mathbf{x}_1(n))\mathbf{H}_0 \\ y(2n-1) &= \mathbf{x}_1(n)\mathbf{H}_0 + \mathbf{x}_0(n-1)\mathbf{H}_1 \\ &= \mathbf{x}_1(n)(\mathbf{H}_0 + \mathbf{H}_1) + (\mathbf{x}_0(n-1) - \mathbf{x}_1(n))\mathbf{H}_1 \end{aligned}$$

Therefore, the calculation of the two terms $y(2n)$ and $y(2n-1)$ requires the calculation of a total of four terms. However, one of them, $\mathbf{x}_1(n)(\mathbf{H}_0 + \mathbf{H}_1)$, appears twice. If P refers to the length of the filter h, the lengths of $\mathbf{H}_0$ and $\mathbf{H}_1$ are at the most equal to $P/2$. Hence the three terms of the calculation of $y(2n)$ and $y(2n-1)$ correspond to $P/2$ length filtering. Figure 5.4 illustrates all these calculations.

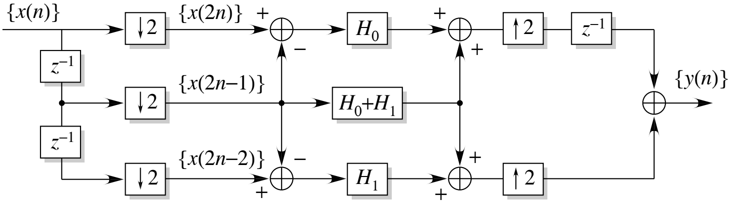


Figure 5.4 – *Block filtering: the case where $M = N = 2$*

To sum up, in order to calculate $y(2n)$ and $y(2n-1)$, the number of MAC operations is roughly $3 \times P/2$. This value should be compared to the $2 \times P$

MAC operations of the direct calculations. You may, as an exercise, simulate the process described in Figure 5.4. There is more than one method organizing the process. Consider for example:

$$\begin{aligned} y(2n) &= \mathbf{x}_0(n)\mathbf{H}_0 + \mathbf{x}_1(n)\mathbf{H}_1 \\ &= (\mathbf{x}_0(n) + \mathbf{x}_1(n))(\mathbf{H}_0 + \mathbf{H}_1) - \mathbf{x}_0(n)\mathbf{H}_1 - \mathbf{x}_1(n)\mathbf{H}_0 \\ y(2n-1) &= \mathbf{x}_1(n)\mathbf{H}_0 + \mathbf{x}_0(n-1)\mathbf{H}_1 \end{aligned}$$

Notice that the term $\mathbf{x}_0(n-1)\mathbf{H}_1$ was calculated previously. Hence there are indeed only three MAC operations at this stage of the calculation. We can also consider parallel block processing for values of M and N different from 2.

5.1.4 FFT filtering

A possibility for accelerating filtering operations is to work in the frequency domain, using Fourier transforms to take advantage of the FFT algorithm's speed. Unfortunately, this is not as simple as it seems, because linear filtering uses a linear convolution:

$$y(n) = \textstyle\sum_{m=-\infty}^{+\infty} x(m)h(n-m) \tag{5.2}$$

the DTFT of which is $H(f)X(f)$, whereas the product of the DFTs is the DFT of the *circular convolution*. As a reminder, here is its expression 14.3:

$$\textstyle\sum_{m=0}^{N-1} x(m)h((n-m) \bmod N) \tag{5.3}$$

A simple calculation shows that expressions 5.2 and 5.3 lead to completely different results. Consider a finite impulse response filter $\{h_N(n)\}$ and a sequence $\{x(n)\}$. The output value at the time n is:

$$y(n) = h_N(0)x(n) + \cdots + h_N(N-1)x(n-N+1) \tag{5.4}$$

For a clearer picture, let us assume $N = 8$. We are going to calculate the terms resulting from a circular convolution of the length 8 block $\{x(n), \ldots, x(n-7)\}$ with a filter with the coefficients $\{h(0), \ldots, h(7)\}$. The following table describes the operations modulo 8.

m	0	1	2	3	4	5	6	7
h_m	h_0	h_1	h_2	h_3	h_4	h_5	h_6	h_7
x	x_{n-7}	x_{n-6}	x_{n-5}	x_{n-4}	x_{n-3}	x_{n-2}	x_{n-1}	x_n
$h_{-m \bmod 8}$	h_0	h_7	h_6	h_5	h_4	h_3	h_2	h_1
$h_{1-m \bmod 8}$	h_1	h_0	h_7	h_6	h_5	h_4	h_3	h_2
				$\vdots$				
$h_{7-m \bmod 8}$	h_7	h_6	h_5	h_4	h_3	h_2	h_1	h_0

Notice that among the 8 results of the circular convolution, only the last one, $h_0x_n + h_1x_{n-1} + \cdots + h_7x_{n-7}$, corresponds to one of the terms from the linear convolution, making this approach completely hopeless. This is actually downright wrong, as we are going to see now.

The overlap-save algorithm

Consider an $N = 5$ length filter with its coefficients $h(0), \ldots, h(4)$ completed by 3 zeros. Let $\{x(n), \ldots, x(n-7)\}$ be the $L = 8$ length input block. As we did before, we can build the sequence of the 8 output values by using the following table:

m	0	1	2	3	4	5	6	7
h_m	h_0	h_1	h_2	h_3	h_4	0	0	0
$\mathbf{x}$	x_{n-7}	x_{n-6}	x_{n-5}	x_{n-4}	x_{n-3}	x_{n-2}	x_{n-1}	x_n
$h_{-m \mod 8}$	h_0	0	0	0	h_4	h_3	h_2	h_1
$h_{1-m \mod 8}$	h_1	h_0	0	0	0	h_4	h_3	h_2
				$\vdots$				
$h_{7-m \mod 8}$	0	0	0	h_4	h_3	h_2	h_1	h_0

The resulting circular convolution outputs are:

$$\begin{aligned}
y_c(0) &= h_0x_{n-7} + h_4x_{n-3} + h_3x_{n-2} + h_2x_{n-1} + h_1x_n \\
y_c(1) &= h_1x_{n-7} + h_0x_{n-6} + h_4x_{n-2} + h_3x_{n-1} + h_2x_n \\
y_c(2) &= h_2x_{n-7} + h_1x_{n-6} + h_0x_{n-5} + h_4x_{n-1} + h_3x_n \\
y_c(3) &= h_3x_{n-7} + h_2x_{n-6} + h_1x_{n-5} + h_0x_{n-4} + h_4x_n \\
y_c(4) &= h_4x_{n-7} + h_3x_{n-6} + h_2x_{n-5} + h_1x_{n-4} + h_0x_{n-3} \\
y_c(5) &= h_4x_{n-6} + h_3x_{n-5} + h_2x_{n-4} + h_1x_{n-3} + h_0x_{n-2} \\
y_c(6) &= h_4x_{n-5} + h_3x_{n-4} + h_2x_{n-3} + h_1x_{n-2} + h_0x_{n-1} \\
y_c(7) &= h_4x_{n-4} + h_3x_{n-3} + h_2x_{n-2} + h_1x_{n-1} + h_0x_n
\end{aligned}$$

Notice that the four last expressions correspond to terms found with the linear convolution:

$$\begin{aligned}
y_c(4) &= y(n-3) \\
y_c(5) &= y(n-2) \\
y_c(6) &= y(n-1) \\
y_c(7) &= y(n)
\end{aligned}$$

In order to calculate the next 4 values, we have to choose $\{x(n+4), \ldots, x(n-3)\}$ as our input block. This block partly overlaps the previous one (see Figure 5.5), hence the word overlap in the name "overlap-save algorithm".

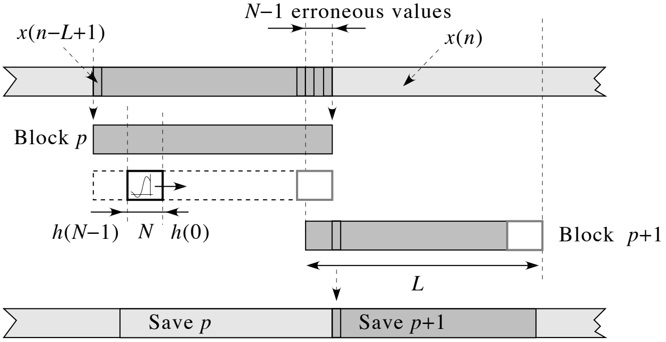


Figure 5.5 – *Overlap-save algorithm*

All of the operations can be summed up as follows:

Overlap-save

1. *Calculation (performed only once) of the DFT of the L length sequence $h(n)$ completed by $(L-N)$ zeros. L (the DFT's length) is usually a power of* 2.
2. *Calculation of the DFT of an L length block extracted from the input data with an overlap of the $(N-1)$ last values of the previous block.*
3. *Term-by-term multiplication of the two DFTs, followed by an IDFT.*
4. *The $(L-N+1)$ terms corresponding to the linear convolution 5.2 are saved.*

Let us compare the number of operations for the *overlap-save* algorithm with that of a direct calculation. The direct calculation of one convolution point for an N length impulse response requires a loop comprising N MAC operations.

Using the overlap-save algorithm, the impulse response's DFT is calculated *in advance*. There are two L length FFTs left for each step (one direct, one inverse) and L complex multiplications, in all a calculation load of roughly $2 \times L\log_2(L) + L = 2L\log_2(L\sqrt{2})$ MAC operations. This calculation provides us with a block of $(L-N+1)$ convolution points, equivalent to a load of about

$2L\log_2(L)/(L-N+1)$ MAC operations per calculation point. Hence the gain is roughly:

$$G(N,L) = \frac{(L-N+1)N}{2L\log_2(L\sqrt{2})}$$

Thus, for $N = L/2$ and $N \geq 32$, the FFT technique is quicker.

Other parameters have to be considered. The FFT calculation implies the use of array pointers, which cause a considerable increase in the calculation time. The FFT also requires memory space to save the data arrays that are too large for the filter's memory. This is why convolution calculations that use the FFT are usually undertaken only with filters with a length of more than a hundred coefficients. In acoustics, impulse response of a quarter-second sampled at 8,000 Hz lead to lengths of 2,000 samples. You also have to add to that the delay caused by block processing, a delay roughly equal the block's length. For some applications, this delay is reason enough to discard these techniques.

Overlap-add algorithm

Consider once again the previous example of a filter $\{h(0), \ldots, h(4)\}$. We still hope to obtain the output:

$$y(n) = h_0 x_n + h_1 x_{n-1} + h_2 x_{n-2} + h_3 x_{n-3} + h_4 x_{n-4}$$

Consider the convolutions concerning two consecutive length 8 blocks labelled $\mathbf{x}_p$ and $\mathbf{x}_{p+1}$.

m	0	1	2	3	4	5	6	7
h_m	h_0	h_1	h_2	h_3	h_4	0	0	0
$\mathbf{x}_p$	x_{n-3}	x_{n-2}	x_{n-1}	x_n	0	0	0	0
$\mathbf{x}_{p+1}$	x_{n+1}	x_{n+2}	x_{n+3}	x_{n+4}	0	0	0	0
$h_{-m \bmod 8}$	h_0	0	0	0	h_4	h_3	h_2	h_1
$h_{1-m \bmod 8}$	h_1	h_0	0	0	0	h_4	h_3	h_2
				$\vdots$				
$h_{7-m \bmod 8}$	0	0	0	h_4	h_3	h_2	h_1	h_0

The values obtained from the first block are:

$$
\begin{aligned}
y_{c,p}(0) &= h_0x_{n-3}\\
y_{c,p}(1) &= h_1x_{n-3} + h_0x_{n-2}\\
y_{c,p}(2) &= h_2x_{n-3} + h_1x_{n-2} + h_0x_{n-1}\\
y_{c,p}(3) &= h_3x_{n-3} + h_2x_{n-2} + h_1x_{n-1} + h_0x_n\\
y_{c,p}(4) &= h_4x_{n-3} + h_3x_{n-2} + h_2x_{n-1} + h_1x_n\\
y_{c,p}(5) &= h_4x_{n-2} + h_3x_{n-1} + h_2x_n\\
y_{c,p}(6) &= h_4x_{n-1} + h_3x_n\\
y_{c,p}(7) &= h_4x_n
\end{aligned}
$$

and the values obtained from the next block are:

$$
\begin{aligned}
y_{c,p+1}(0) &= h_0x_{n+1}\\
y_{c,p+1}(1) &= h_1x_{n+1} + h_0x_{n+2}\\
y_{c,p+1}(2) &= h_2x_{n+1} + h_1x_{n+2} + h_0x_{n+3}\\
y_{c,p+1}(3) &= h_3x_{n+1} + h_2x_{n+2} + h_1x_{n+3} + h_0x_{n+4}\\
y_{c,p+1}(4) &= h_4x_{n+1} + h_3x_{n+2} + h_2x_{n+3} + h_1x_{n+4}\\
y_{c,p+1}(5) &= h_4x_{n+2} + h_3x_{n+3} + h_2x_{n+4}\\
y_{c,p+1}(6) &= h_4x_{n+3} + h_3x_{n+4}\\
y_{c,p+1}(7) &= h_4x_{n+4}
\end{aligned}
$$

As you can see:

$$
\begin{aligned}
y(n+1) &= y_{c,p}(4) + y_{c,p+1}(0)\\
y(n+2) &= y_{c,p}(5) + y_{c,p+1}(1)\\
y(n+3) &= y_{c,p}(6) + y_{c,p+1}(2)\\
y(n+4) &= y_{c,p}(7) + y_{c,p+1}(3)
\end{aligned}
$$

The conclusion is that if the input block is completed with $N-1=4$ zeros, the circular convolution will calculate incomplete sums. These sums will then be completed with values obtained from the next block translated by $N-1$ values. The sequence of operations can be summed up in the following way:

Overlap-add

1. *Calculation of the DFT of the N length sequence $h(n)$ completed with $(L-N)$ zeros. Usually the length L of the DFT is 2^P.*
2. *Calculation of the DFT of a length $(L-N+1)$ block extracted from the input data without any overlap and completed with $(N-1)$ zeros.*
3. *Term-by-term multiplication of the two DFTs, followed by an IDFT;*
4. *Sum of the current block and of the next block with an overlap of $(N-1)$ values.*

You can check that the overlap algorithm leads to basically the same calculation load as the overlap-save algorithm.

To sum everything up, the overlap-save performs an overlap on the inputs then delivers the result, whereas the overlap-add technique performs an overlap not on the input but on the output (see Figure 5.6).

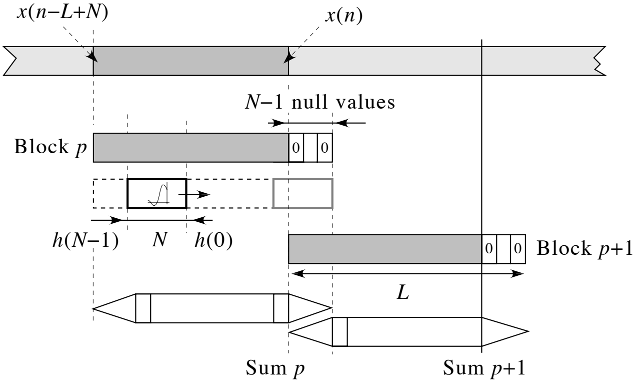


Figure 5.6 – *Overlap-add algorithm*

Exercise 5.3 (FFT filtering)
Let $x(n)$ be a signal such that $x(n) = \sin(2\pi f_0 n) + \sin(2\pi f_1 n)$, where $f_0 = 0.15$ and $f_1 = 0.3$ and let $h(n)$ be the following impulse response filter:
`h(n)=[0.0002 0.0134 0.0689 0.1676 0.2498 0.2498 0.1676 0.0689 0.0134 0.0002]`.

1. Normalize the filter's coefficients so as to have the gain at the frequency 0 equal to 1.

2. Display on the same graph the original signal and the filtered signal obtained with the `filter` function.

3. Display the filter's complex gain and the spectra of the original signal and of the filtered signal.

4. Perform the filtering using an FFT on the entire signal.

5. Perform the same process with length 32 blocks. Notice that this requires an overlap of consecutive blocks.

Appendix A6 gives another approach of the FFT filtering based on the properties of circulant matrices.

5.2 Filter banks

The idea of using several parallel filters to "simultaneously" analyze several frequency bands is very old. That is how some analog spectrum analyzers work. Several filters, forming what is called a "filter bank", with slightly overlapping frequency responses, cover the entire extent of the frequency band we wish to analyze. The short term Fourier transform time-frequency analysis (see paragraph 3.2) is another example.

Signal spectrum analysis is not the only application of filter banks. For the purpose of processing improvements, we can imagine performing operations on signals coming from different filters. This is what is represented in Figure 5.7.

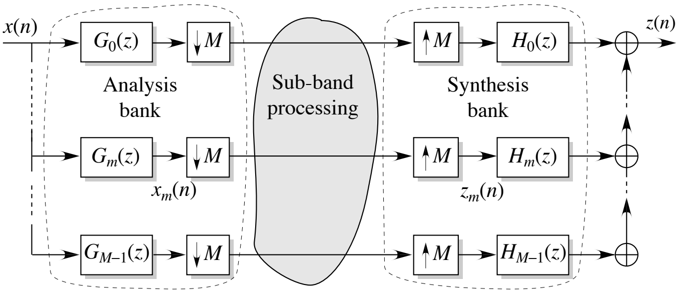


Figure 5.7 – *How filter banks work*

There are two advantages to this method: on one hand, the calculations are parallel, and on the other hand, the processes can be adapted to the various channels. Among the main applications of subband filtering techniques, a few are worth mentioning, such as subband coding, multicarrier modulations, analog-to-digital conversion (Σ-Δ), etc. In this paragraph, we will only present some results concerning filter banks that can be encountered when dealing with processing architectures.

In digital processing, the fact that each channel operates on narrower frequency bands allows the possibility, according to the sampling theorem, of reducing the sampling rate at the filter's output, as it is shown in Figure 5.7. We end up with a system for which different processing frequencies are used simultaneously in different points of the calculation chain. The oversampling and undersampling operations are examples. Two operations form the basis of these techniques: *decimation and expansion*. We have already encountered them in paragraph 4.8, and we will now discuss them in greater detail.

5.2.1 Decimation and expansion

Decimation: an operation that takes one out of every M samples. Symbolically, it is represented by an arrow pointing down (Figure 5.8). According to 4.40:

$$X_{\downarrow M}(z) = \frac{1}{M}\sum_{k=0}^{M-1} X(z^{1/M}W_M^k) \text{ where } W_M = \exp(-2j\pi/M) \quad (5.5)$$

Expansion: an operation that inserts $M-1$ zeros between two samples of the original sequence. Symbolically, it is represented by an arrow pointing up (Figure 5.9). According to 4.38:

$$X_{\uparrow M}(z) = X(z^M) \quad (5.6)$$

Property 5.1 (Filtering and decimation)
We have the property illustrated by Figure 5.8.

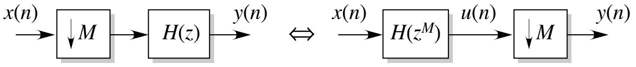


Figure 5.8 – *Equivalence implicating a filtering and a decimation*

HINT: on the right-hand side of Figure 5.8, we have:

$$Y(z) = U_{\downarrow M}(z) = \frac{1}{M}\sum_{n=0}^{M-1} U(z^{1/M}W_M^n)$$

By choosing $U(z) = H(z^M)X(z)$, we get:

$$\begin{aligned} Y(z) &= \frac{1}{M}\sum_{n=0}^{M-1} X(z^{1/M}W_M^{-n})H[(z^{1/M}W_M^{-n})^M] \\ &= \sum_{n=0}^{M-1} X(z^{1/M}W_M^{-n})H(z) = H(z)X_{\downarrow M}(z) \end{aligned}$$

which corresponds to the expression for the process on the left-hand side of Figure 5.8. ■

Property 5.2 (Filtering and expansion)
We have the properties illustrated by Figure 5.9.

HINT: starting off with the diagram on the left of Figure 5.9, and with $U(z) = H(z)X(z)$, we have:

$$Y(z) = U(z^M) = H(z^M)X(z^M) = H(z^M)X_{\uparrow M}(z)$$

corresponding to the expression for the process on the right of Figure 5.9. ■

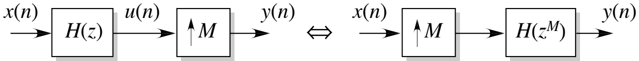


Figure 5.9 – *Equivalence implicating a filtering and an expansion*

Application: the comb decimation filter

Consider the filter represented in Figure 5.10. It is composed of the filter with the transfer function $1/(1 - z^{-1})$ cascaded with the filter with the transfer function $(1 - z^{-M})$.

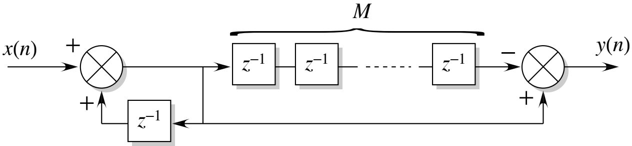


Figure 5.10 – *Comb filter composed of the filter* $1/(1 - z^{-1})$ *followed by the filter* $(1 - z^{-M})$

Hence its transfer function has the expression:

$$H_z(z) = \frac{1}{1 - z^{-1}} \times (1 - z^{-M})$$

A simplification leads to $H_z(z) = 1 + z^{-1} + \cdots + z^{-(M-1)}$. This is therefore an FIR filter with the impulse response $h(n) = 1$ for $n \in \{0, \ldots, (M-1)\}$ and 0 otherwise. Notice that $1/(1 - z^{-1})$ has a pole in $z = 1$ (zero frequency) and that $(1 - z^{-M})$ has M zeros placed on the unit circle in $W_m = e^{2j\pi m/M}$ where $m \in \{0, \ldots, (M-1)\}$.

Because of the location of the zeros of $(1 - z^{-M})$, regularly spread out on the unit circle, the filter is called a *comb filter*. After a simplification, $H_z(z)$ has $(M-1)$ zeros in $e^{2j\pi m/M}$ where $m \in \{1, \ldots, (M-1)\}$, and no poles.

The frequency response of the filter $H_z(z)$ is represented in Figure 5.11 for $M = 16$. It has a main lobe centered at 0 with a width of $2/M$. It is therefore a low-pass filter. By placing the first cell's pole on another of the second cell's zeros, we can obtain a low-pass filter (see exercise 5.4).

Exercise 5.4 (Band-pass filter based on a comb filter)
In a method similar to the one used for the low-pass comb filter in Figure 5.10, design a real band-pass filter centered at the frequency $f_m = m/M$.

Although the filter in Figure 5.11 is, after simplification, an FIR filter that can therefore be achieved by a stable structure, the diagram in Figure 5.10 is unstable because the first cell can produce an unbounded output even if the

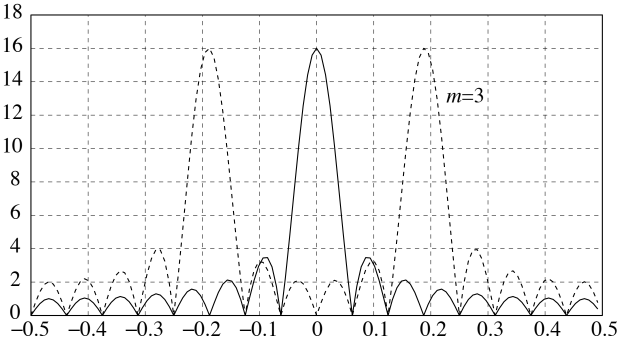


Figure 5.11 – *Frequency response of a comb filter for $M = 16$*

input is bounded. For systems that use comb filters (see Figure 5.14), it is ensured that this never happens. We will now see an application of the comb filter.

The comb filter represented in Figure 5.10 can be used for designing a low-pass filter (but not a very selective one) in the undersampling operation (see paragraph 4.8.2, page 155), by placing it before an M order decimator. The result is the first system represented in Figure 5.12.

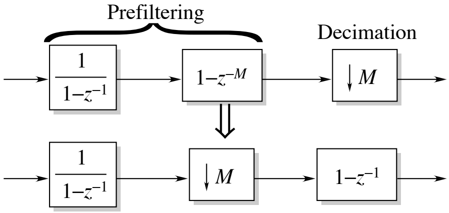


Figure 5.12 – *Permutation of the decimator and the derivative filter*

According to property 5.1, the filtering and decimation operations can be performed in any order. We end up with the second system in Figure 5.12. In this system $I(z) = 1/(1 - z^{-1})$ performs the operation that associates the output $u(n)$ with the input $x(n)$ as follows:

$$u(n) = u(n-1) + x(n)$$

which is an accumulation/integration and $D(z) = 1 - z^{-1}$ performs the operation that associates the output $y(n)$ with the input $v(n)$ as follows:

$$y(n) = v(n) - v(n-1)$$

which can be seen as the approximation of a derivative filter.

The selectivity can be increased by cascading several integrators and several derivative filters, as it is done on certain audio CD players using what is called the *one-bit stream* technique. Figure 5.13 explains how it works.

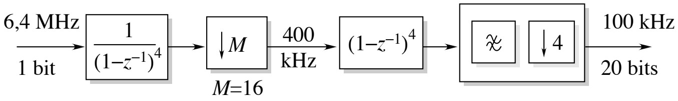


Figure 5.13 – *Undersampling filter*

Starting off with a binary flux at 6.4 MHz, we accumulate input values, 0 or 1, then we decimate by a factor of 4. We obtain a flux of values represented on 20 bits and sampled at 400 kHz. As we have already said, the integrator cascade is, by nature, unstable. However, it can be shown that if we use a modulo M summer, the "integrator, decimator, derivative filter" set does not cause any overflow, so long as the summer contains M bits. Because the cascade is comprised of four of these systems, an $M + 4 = 20$ bits summer is used. Finally, the calculations performed by the fourth band output filter are processed with 38 bits.

5.2.2 Filter banks

An analysis filter bank is a group of parallel digital filters, the input signals of which are $x(n)$, that cuts up the frequency band in K subbands. The synthesis filter bank is a group of K filters placed after the the analysis filter bank and generating the signal $\hat{x}(n)$.

The processing system can be represented by a group of filters connected by *undersampling* and *expansion* operators as shown in Figure 5.14.

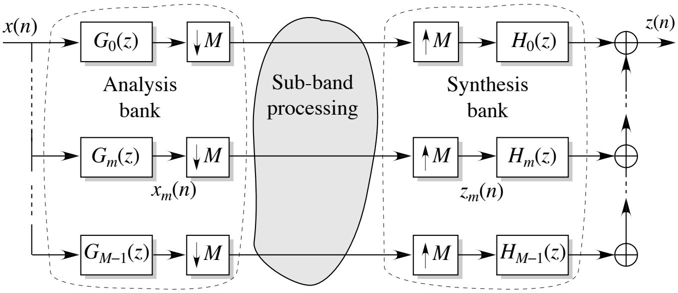


Figure 5.14 – *How the filter bank works*

We now reconsider the problem of *perfect reconstruction*: is there a filter bank such that the aliasing effects for each band compensate each other exactly on the entire band? This question is justified by the fact that if no processing is done, the least that can be expected of this structure is to produce an output signal identical to the input signal. To make the rest of this discussion simpler, we will consider the case where $M = 2$ according to Figure 5.15, a case of important practical use.

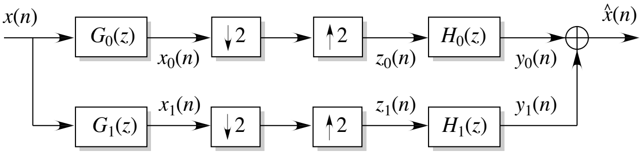


Figure 5.15 – *Two channel filter bank*

Obvious solutions

Note that the perfect reconstruction problem has at least two obvious solutions. The first one consists of taking two ideal low-pass filters in the $(0, 1/4)$ band for $G_0(z)$ $H_0(z)$, and two ideal high-pass filters in the $(1/4, 1/2)$ band for $G_1(z)$ $H_1(z)$. This solution has the advantage of using very selective filters, but its filters are infinite impulse response filters, which is a drawback.

The second solution simply consists of choosing $G_0(z) = H_1(z) = 1$ and $G_1(z) = H_0(z) = z^{-1}$. In this case, the analysis filter bank is merely a demultiplexer distributing the even index values of $x(n)$ to one channel and the odd index values to the other channel, while the synthesis filter bank is a multiplexer that interlaces the two channels. This solution uses FIR filters (with only one coefficient!). However, these filters are unfortunately band-pass filters with a gain of 1 without any frequency selectivity.

One of the major problems we are faced with when using filter banks is the difficulty of finding a solution that has a finite impulse response, a good selectivity and the ability to perform perfect reconstruction, all at once.

Perfect reconstruction equations

By referring to the diagram in Figure 5.15, and by using formulae 5.5 and 5.6, we can write:

$$\begin{aligned} Z_0(z) = (G_0X)_{\downarrow 2\uparrow 2} &= [G_0(z)X(z) + G_0(-z)X(-z)]/2 \\ Z_1(z) = (G_1X)_{\downarrow 2\uparrow 2} &= [G_1(z)X(z) + G_1(-z)X(-z)]/2 \end{aligned}$$

and:

$$\begin{aligned} Y_0(z) &= H_0(z)[G_0(z)X(z) + G_0(-z)X(-z)]/2 \\ Y_1(z) &= H_1(z)[G_1(z)X(z) + G_1(-z)X(-z)]/2 \end{aligned}$$

This leads us to the reconstructed sequence $\hat{x}(n)$ by $\hat{X}(z) = Y_0(z) + Y_1(z)$ and therefore:

$$\begin{aligned} 2\hat{X}(z) &= \left[G_0(z)H_0(z) + G_1(z)H_1(z)\right] X(z) + \\ &\quad \left[G_0(-z)H_0(z) + G_1(-z)H_1(z)\right] X(-z) \\ &= T(z)X(z) + A(z)X(-z) \end{aligned}$$

Perfect reconstruction is ensured when $\hat{X}(z) = z^{-r}X(z)$. This gives us the following two conditions:

$$T(z) = G_0(z)H_0(z) + G_1(z)H_1(z) = 2z^{-r} \tag{5.7}$$

$$A(z) = G_0(-z)H_0(z) + G_1(-z)H_1(z) = 0 \tag{5.8}$$

The first condition (5.7) expresses the absence of distortion, due to the fact that the transfer function $T(z)$ has a gain equal to 1 and a linear phase. The second one (5.8) ensures that the term $X(-z)$, characterizing the spectrum aliasing, is zeroed out.

Quadrature filters

A first solution consists of imposing:

$$H_0(z) = G_1(-z) \text{ and } H_1(z) = -G_0(-z) \tag{5.9}$$

which ensures condition 5.8. Condition 5.7 then becomes:

$$H_0(z)G_0(z) - G_0(-z)H_0(-z) = 2z^{-r} \tag{5.10}$$

If $H_0(z)$ and $G_0(z)$ are two polynomials in z^{-1}, equation 5.10 can be expressed $\alpha(z) - \alpha(-z) = 2z^{-r}$, where we have defined $\alpha(z) = H_0(z)G_0(z)$. In $\alpha(z) - \alpha(-z)$, only the odd degree coefficients of $\alpha(z)$ remain. Therefore, all the odd degree coefficients of $\alpha(z)$ must be equal to 0, except for the r-th degree coefficient. This implies, incidentally, that r is odd. We then have to factorize $\alpha(z) = H_0(z)G_0(z)$. Because the two constraints 5.9 are supposed to be obeyed at all times, two simple solutions arise:

– We impose $G_0(z) = G_1(-z)$. If we change over to the DTFTs, we get $G_0(e^{2j\pi f}) = G_1(e^{2j\pi(f-1/2)})$. Because the filters are real, this expression implies that the frequency responses of the filters have a "mirror" symmetry about the frequency 1/4. This is called a *QMF filter bank*, short for *Quadrature Mirror Filters*. Unfortunately, there are very few solutions, and they are not selective. In order to show this, we replace $G_0(z) = G_1(-z)$ in the first expression of 5.9, meaning $H_0(z) = G_1(-z)$, and we get $G_0(z) = H_0(z)$. Replacing it in 5.10 leads us to:

$$H_0^2(z) - H_0^2(-z) = 2z^{-r}$$

For this solution to be satisfied, $H_0(z)$ cannot have more than two non-zero coefficients, in other words $H_0(z) = h_0 z^{-k_0} + h_1 z^{-k_1}$. If we identify the terms, we get:

$$4h_0 h_1 z^{-(k_0+k_1)} = 2z^{-r}$$

and therefore k_0 and k_1 can have any value so long as the sum is odd, for example, $k_0 = 0$ and $k_1 = 1$, and $h_0 h_1 = 1/2$. By imposing that the filters be linear-phase filters, and therefore $h_0 = h_1$, we get $h_0 = h_1 = 1/\sqrt{2}$. This result is not satisfactory because the obtained filters are very poorly selective. Thus, the frequency response of $H_0(z)$, for $k_0 = 0$ and $k_1 = 1$, is $|H_0(e^{2j\pi f})|^2 = \cos^2(\pi f)/2$.

– The condition $G_1(z) = G_0(-z)$ is now replaced by $G_1(z) = (-z)^{-N} G_0(-z^{-1})$ or $g_1(n) = (-1)^n g_0(N-n)$, which is equivalent. This is called a *CQF filter bank*, short for *Conjugate Quadrature Filters*. By replacing $G_1(z) = (-z)^{-N} G_0(-z^{-1})$ in 5.9, that is to say $H_0(z) = G_1(-z)$, we have $H_0(z) = (-z)^{-N} G_0(z^{-1})$. Replacing it in 5.10 leads us to:

$$z^{-N}\left(G_0(z)G_0(z^{-1}) + G_0(-z)G_0(-z^{-1})\right)$$

and a sufficient condition on the phase is provided by:

$$G_0(z)G_0(z^{-1}) + G_0(-z)G_0(-z^{-1}) = 1$$

The transfer function $D(z) = G_0(z)G_0(z^{-1})$ is sometimes called a *zero-phase half-band*. Because $G_0(e^{2j\pi f})$ satisfies $|G_0(e^{2j\pi f})|^2 + |G_0(e^{2j\pi(f-1/2)}|^2 = 1$, G_0 is said to be "power symmetric". Searching for a solution can be summed up as follows:

Steps*:*

1. *Find an odd order, "power symmetric" filter $D(z)$, approximately half-band.*

   In order to do this, we can start with the window method (using a triangular window for which positivity is ensured), or with an iterative method such as the Parks-McClellan algorithm [9] (the problem is that there is no guarantee that the phase will be linear).
2. *Perform a spectral decomposition of $D(z)$ in $G_0(z)G_0(z^{-1})$.*
3. *Construct the filter bank using $G_1(z) = (-z)^{-N} G_0(-z^{-1})$, then:*

   $$H_0(z) = G_1(-z) \;\; H_1(z) = -G_0(-z)$$

Orthogonal filters

We will again use equations 5.7 and 5.8, written below:

$$\begin{cases} G_0(z)H_0(z) + G_1(z)H_1(z) = 2z^{-r} \\ G_0(-z)H_0(z) + G_1(-z)H_1(z) = 0 \end{cases}$$

and solve this linear system in order to determine the expressions of $H_0(z)$ and $H_1(z)$ as an expression of $G_0(z)$ and $G_1(z)$. We get:

$$\begin{aligned} H_0(z) &= \frac{2z^{-r}G_1(-z)}{G_0(z)G_1(-z) - G_1(z)G_0(-z)} \\ H_1(z) &= -\frac{2z^{-r}G_0(-z)}{G_0(z)G_1(-z) - G_1(z)G_0(-z)} \end{aligned} \tag{5.11}$$

Property 5.3 *For the two channel filter bank, the perfect reconstruction property is obtained if and only if:*

$$\begin{cases} \sum_k g_0(k)h_0(2n-k) = \delta(2n-r) \\ \sum_k g_1(k)h_1(2n-k) = \delta(2n-r) \\ \sum_k g_1(k)h_0(2n-k) = 0 \end{cases}$$

HINT: let $P(z) = H_0(z)G_0(z)$. Using 5.11, we have:

$$P(z) = 2z^{-r}G_1(-z)G_0(z)/D(z)$$

where $D(z)$ refers to the denominator of $H_0(z)$ in 5.11. Likewise:

$$H_1(z)G_1(z) = -2z^{-r}G_0(-z)G_1(z)/D(z)$$

Because $D(z) = -D(-z)$, we have $H_1(z)G_1(z) = P(-z)$, and we can write:

$$P(z) + P(-z) = 2z^{-r}$$

This condition implies that r is even, and that $p(2n) = \delta(2n-r)$. By noticing that $P(z) = H_0(z)G_0(z)$ is the z-transform of the convolution product of $h_0(n)$ with $g_0(n)$, we get:

$$\sum_k g_0(k)h_0(2n-k) = \delta(2n-r)$$

Now let $Q_0(z) = H_1(z)G_0(z)$:

$$Q_0(z) = -2z^{-r}G_0(z)G_0(-z)/D(z)$$

Because r is even, $Q_0(z)$ is odd. And hence $q_0(2n) = 0$. By noticing that $Q_0(z) = H_1(z)G_0(z)$ is the z-transform of the convolution of $h_1(n)$ with $g_0(n)$, we get $\sum_k g_1(k)h_0(2n-k) = 0$. ■

The sequences $g_0(n)$ $g_1(n)$ on one hand, and $h_0(n)$ and $h_1(n)$ on the other, lead to the definition of two sets of orthogonal sequences. Let:

$$\phi_{2n}(k) = g_0(2n-k), \quad \phi_{2n+1}(k) = g_1(2n-k)$$
$$\psi_{2n}(k) = h_0(k-2n), \quad \psi_{2n+1}(k) = h_1(k-2n)$$

Property 5.3 shows that the two sequences $\{\phi_n(k)\}$ and $\{\psi_n(k)\}$ verify for any $n \neq n'$:

$$\sum_k \phi_n(k)\psi_{n'}(k) = 0$$

The two sets $\{\phi(n)\}$ and $\{\psi(n)\}$ are said to have the *bi-orthogonality property*. This is the equivalent for infinite dimension of the property of two matrices such that $\Psi^T\Phi = \text{diag}(d_1,\dots,d_K)$ where $\text{diag}(d_1,\dots,d_K)$ is a diagonal matrix, the identity being a particular case.

We will now discuss the orthogonal case [98], where the sequences $\phi_n(k)$ and $\psi_n(k)$ coincide, that is where one is equal to the other translated. A sufficient condition is to have $h_0(n) = g_0(r-n)$ and $h_1(n) = g_1(r-n)$. Hence perfect reconstruction and orthogonality require the impulse responses of the synthesis filters to be reversed copies of the impulse responses of the analysis filters. Changing over to the z-transforms, this leads to:

$$H_i(z) = z^{-r}G_i(1/z) \text{ where } i = \{0,1\} \tag{5.12}$$

This means, first of all, that $P(z)$, defined by $P(z) = H_0(z)G_0(z)$ can be written:

$$P(z) = z^{-r}G_0(1/z)G_0(z)$$

This relation implies that if z_0 is a root of $P(z)$, then $1/z_0$ is also a root of $P(z)$. Hence, the roots of $P(z)$ are pairs of inverse values, one inside and one outside the unit circle.

By replacing 5.12 in the second equation of 5.11, we get:

$$G_0(-1/z)G_0(z) + G_1(-1/z)G_1(z) = 0$$

If the polynomials $G_0(z)$ and $G_1(z)$ share the same finite degree (FIR filters of the same length) and are different from one another, then the roots of $G_1(z)$ have to be roots of $G_0(-1/z)$. Therefore, $G_1(z) = -z^{2K-1}G_0(-1/z)$. This relation can be expressed, in the temporal domain, as:

$$g_1(n) = (-1)^n g_0(2K-1-n)$$

To sum up, calculating analysis filter banks using orthogonal filters is achieved using the following method: starting off with $P(z) = G_0(z)G_0(1/z)$ which verifies $P(z) + P(-z) = 2$:

- we associate with $G_0(z)$ the roots of $P(z)$ that are inside the unit circle, then we calculate $g_0(n)$;
- we calculate $h_0(n) = g_0(-n)$;
- we calculate $g_1(n) = (-1)^n g_0(2K - 1 - n)$;
- we calculate $h_1(n) = g_1(-n)$.

We still have to find a function $P(z) = G_0(z)G_0(1/z)$ such that $P(z) + P(-z) = 2$.

A first crude method consists of imposing the relation $P(z)+P(-z) = 2$ by choosing a sequence $p(n)$ such that $p(2n) = 0$ in the following manner: we start off with an even sequence w_n, for example, the one obtained by the FIR filter design method (window method, Remez method), and all the even index terms are replaced by zero, except for the zero index term. This can be expressed as follows:

$$p(n) = w_n c_n$$

where $c_{2n} = \delta(n)$. However, this does not guarantee that $P(z)$ can be expressed as $G_0(z)G_0(1/z)$, or that $P(e^{2j\pi f})$ is positive, which is equivalent. We can then determine the sequence c_n such that $P(e^{2j\pi f}) > 0$. As a consequence, the relation $P(z) + P(-z) = 2$ is not quite true anymore, and becomes even less true as the minimum negative value of the DTFT of w_n becomes smaller.

Let us now see an important example related to the Daubechies wavelets. We start with a polynomial $P(z)$, such that it is at the frequency $1/2$. As a consequence, this introduces in the sequence $p(n)$ a kind of regularity similar to the signal smoothing property when the energy of the high frequencies is reduced, hence the idea to place a great number of zeros in $z = -1$. For this we assume:

$$P(z) = (1 + z)^k (1 + z^{-1})^k R(z)$$

where $R(z)$ can be expressed as $R_1(z)R_1(1/z)$. $R(z)$ is therefore a symmetrical polynomial for which the degrees of its terms vary from $-s$ to $+s$. Therefore, $P(z)$ has $2k + 2s$ roots and is dependent on $2k + 2s + 1$ coefficients, $(k + s)$ of which have to be equal to zero ($p(2n) = 2\delta(k)$). This leads to $(k+s)$ equations. Yet we have $2s + 1$ linearly independent coefficients in $R(z)$. This means we have to set $k + s = 2s + 1$, or $s = k - 1$. Thus, for $k = 2$, we get $s = 1$. Hence the length of the filter $G_0(z)$ is 4. Generally speaking, this method leads to FIR filters with lengths of $L = 2k$.

Let us calculate the coefficients for $k = 2$. For this we assume $R(z) = (\alpha z + \beta + \alpha z^{-1})$. The expression of the condition $P(z) + P(-z) = 2$ will give us two equations with two unknowns α and β. First we have:

$$\begin{aligned} P(z) &= \alpha z^{-3} + (4\alpha+\beta)z^{-2} + (4\beta+7\alpha)z^{-1} + (8\alpha+6\beta) \\ &\quad +(4\beta+7\alpha)z + (4\alpha+\beta)z^{2} + \alpha z^{3} \end{aligned}$$

The condition:

$$P(z)+P(-z) = 2((4\alpha+\beta)z^{-2} + (8\alpha+6\beta) + (4\alpha+\beta)z^{2}) = 2$$

is met if $4\alpha+\beta=0$ and $8\alpha+6\beta=1$. This leads to $\alpha=-1/16$ and $\beta=1/4$. If we factorize $R(z)$, then associate with $G_0(z)$ the roots inside the unit circle, we get:

$$G_0(z) = \frac{1}{4\sqrt{2}}\left((1+\sqrt{3}) + (3+\sqrt{3})z^{-1} + (3-\sqrt{3})z^{-2} + (1-\sqrt{3})z^{-3}\right)$$

This leads to $h_0(n) = g_0(-n)$, then $g_1(n) = (-1)^n g_0(3-n)$ and $h_1(n) = g_1(-n)$.

The following program calculates the coefficients of $G_0(z)$, plots the gains of the analysis filters, and checks the perfect reconstruction property on a trajectory.

```
%===== DAUB4.M
clear
r=4;       % Delay due to the bank
g0=[1+sqrt(3);3+sqrt(3);3-sqrt(3);1-sqrt(3)]/4/sqrt(2);
h0=g0(r:-1:1); g1=-h0 .* ((-1) .^ (0:r-1)');
h1=g1(r:-1:1);
%===== Gains
Lfft=1024; freq=[0:Lfft-1]/Lfft;
G0f=abs(fft(h0,Lfft)); G1f=abs(fft(h1,Lfft));
subplot(311); plot(freq,[G0f G1f]); grid;
set(gca,'Xlim',[0 .5])
%===== Verification
N=1000; x=randn(N,1);
%===== Analysis
x0=filter(g0,1,x); x1=filter(g1,1,x);
%===== Decimation/expansion
v0=x0; v0(1:2:N)=zeros(N/2,1);
v1=x1; v1(1:2:N)=zeros(N/2,1);
%===== Synthesis
y0=filter(h0,1,v0); y1=filter(h1,1,v1);
xchap=y0+y1; max(abs(xchap(r:N)-x(1:N-r+1)))
subplot(312); plot(x(100:120)); grid
subplot(313); plot(xchap(100+r-1:120+r-1)); grid
```

Comments

- We often only restrict ourselves to the two-branch symmetrical filter, because the same segmentation can be applied to both branches (Figure 5.16).

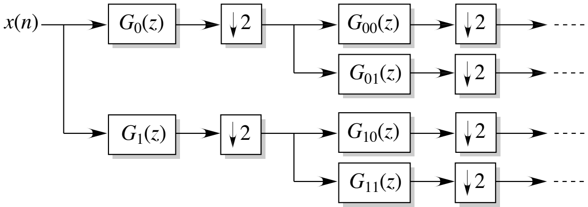


Figure 5.16 – *Decomposition of each branch*

A particular decomposition in *octaves* (Figure 5.17) can be associated with the *wavelets* using multi-scale analysis.

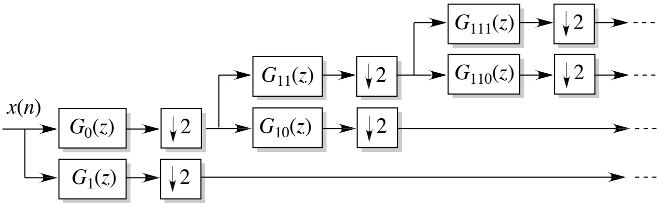


Figure 5.17 – *Decomposition in octaves*

– A commonly used approach in sub-band decomposition techniques uses the FFT calculation structure. Analysis and synthesis filter banks consist of inverse and direct "FFT blocks". A reader curious for more information on this method should read [27, 96].

This page intentionally left blank

Chapter 6

An Introduction to Image Processing

This chapter provides the reader with a few elements on image processing with MATLAB®, which comes equipped with 2D (two dimension) functions, necessary when working in this field, a field not too different from 1D signal processing.

This chapter is merely an introduction. The reader can benefit from reading [31], a rather extensive overview of what is done with images, both still and animated. Some important problems, related to sampling, rectangular, hexagonal or of another kind, to perception, to content aspects in terms of objects, etc., will not be discussed here. The only thing we will be dealing with is handling two dimension arrays. We will also explain how to program some of the functions contained in the "image" *toolbox.*

Examples in this chapter are illustrated by figures that cannot perfectly render the phenomena we are trying to underline. The printing process, whether monochrome or not, adds its own imperfections (quantization, weaving, number of colors, color transcription, etc.) when rendering images. In fact, every part of the digital processing chain, from the data recording device to the printer, has a role that will not be covered in this book.

6.1 Introduction

6.1.1 Image display, color palette

From now on, an image will be considered as a set of *pixels* (the contraction of *picture element*), associated with a rectangular grid of the original image (Figure 6.1).

In MATLAB®, there are several ways to display an image:

- Either directly with an $(N \times M \times 3)$ or $(N \times M \times 4)$ array depending on the color model: RGB (**R**ed, **G**reen and **B**lue), CMYK (**C**yan, **M**agenta, **Y**ellow and blac**K**), HSL (**H**ue, **S**aturation and **L**ightness), CIE Lab

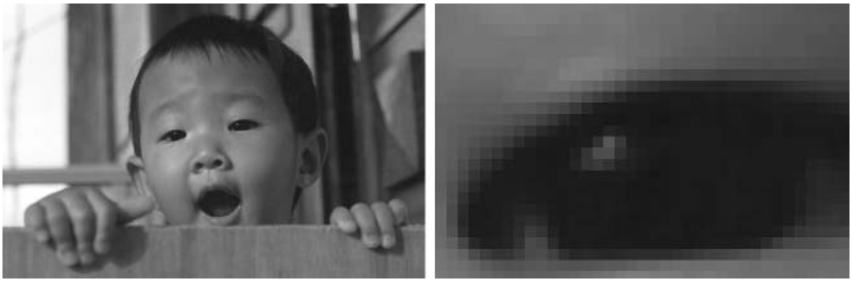


Figure 6.1 – *Each point of the original image has an 8-bit coded "gray-level". Each pixel appears as a gray square*

("Commission Internationale de l'Eclairage": L is for luminance, and a and b are color component coordinates), etc.

In the following example, an image in JPEG format is imported with the use of the **imread** function as a 3 dimension 800 × 580 × 3 array, the 3 indicating that there are three RGB color planes. Notice that the data type used is the *8-bit unsigned integer*:

```
>> xx=imread('elido72.jpg','jpeg');
>> whos
  Name      Size                    Bytes  Class

  ans        1x94                     188  char array
  xx       800x580x3              1392000  uint8 array
```

- either by using a 2D (short for 2 dimension) array and a color palette. This is the display mode we will be using; it is called an *indexed representation*.

Let $\mathbf{A} = [a(i,j)]$, with $1 \le i \le N$ and $1 \le j \le M$, be an $N \times M$ array. The number $a(i,j)$, placed in line i and column j, indicates the color of the point with coordinates (i,j) in the image after it has been "sampled" and "quantified". The line index i represents the horizontal position, and the column index j represents the vertical position. The point with the coordinates $(1,1)$ is placed in the top-left corner (see Figure 6.2).

Example 6.1 (Pixelizing an image) Type:

```
image1=[32 0 48;0 16 0];
image(image1); colormap('gray')
```

The image displayed is comprised of 6 points, or *logical pixels*, and the one associated with `image1(1,1)` is the one in the top-left corner (Figure 6.2).

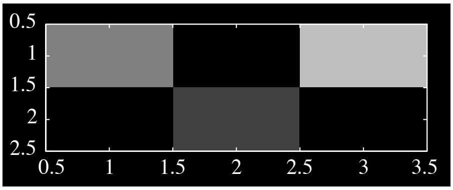


Figure 6.2 – *Six logical pixels: notice the integer x- and y-coordinate corresponding to the "center" of each pixel*

Notice that an element of the array with the index (i, j) can be associated with *several* physical pixels of the display window. In fact, there is no reason for the number of values of the matrix of elements $a(i, j)$ to be equal to the number of *physical pixels* of the display window. Hence, a point with the coordinates (i, j) can be represented by several *physical pixels*, just as a *physical pixel* can be used to represent several points with the coordinates (i, j). From now on, when we use the word *pixel*, we mean a *logical pixel*, that is to say elements identified by the pair (i, j).

If we want to display a Figure and preserve its real size (one screen pixel corresponding to one image pixel), we will be using the properties `units`, `Position`, `AspectRatio`... (these parameters can change from one MATLAB®version to the next). In example 6.1, a real-size display is achieved by typing:

```
set(gca,'units','pixels','Position',[20 20 fliplr(size(image1))])
```

In the indexed representation, $a(i, j)$ indexes a color array called the *palette* (Figure 6.3). The color palette is a $(P \times 3)$ array where each line is used to code a color according to its *Red*, *Green* and *Blue* components (*RGB*) using a real number between 0 and 1.

This representation is convenient since most bitmap editing programs can provide an image description in three planes, each one of them corresponding to a primary color R, G or B, encoded as an integer between 0 and $2^n - 1$ (n-bit encoding). The images we will be considering will be "in levels of gray". MATLAB® has a default palette that can be activated using the `colormap('gray')` instruction.

Type `colormap('gray')` then `colormap`. You get a (64×3) array with three identical columns of values between 0 and 1:

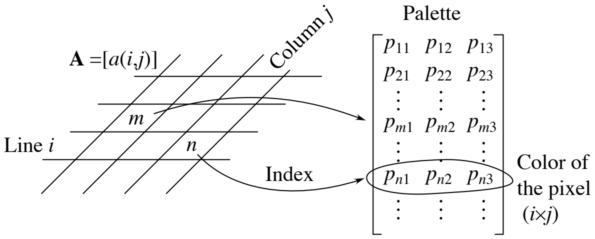

Palette
A =[a(i,j)]
Column j
m
Line i
n
Index
p_{11} p_{12} p_{13}
p_{21} p_{22} p_{23}
p_{m1} p_{m2} p_{m3}
p_{n1} p_{n2} p_{n3}
Color of the pixel (i×j)

Figure 6.3 – *Connection between the image array and the palette*

```
ans =
         0         0         0
    0.0159    0.0159    0.0159
    0.0317    0.0317    0.0317
    0.0476    0.0476    0.0476
    ...
    ...
    ...
    0.9841    0.9841    0.9841
    1.0000    1.0000    1.0000
```

COMMENTS:

- In example 6.1 the zero values of the `image1` array are redefined as 1 and therefore index the color $(0, 0, 0)$, which is black (Figure 6.2).

- Help for the commands `image`, `imagesc` and `colormap` should particularly be looked into.

- The standard palette is constructed linearly. Each column is of the type `[0:1/63:1]'` ($1/63 \simeq 0.0159$). This does not quite correspond to the perception we have of brightness. The visual response is roughly proportional to the logarithm of the intensity (*Fechner-Weber law*), hence the progression of the levels of gray should correspond to this law. In practice, the palette's linear conformation makes our work much easier since palette index lines and gray levels are related by an affine relation.

- Other palettes come standard in the basic version of MATLAB® to make the user's work easier. Use the `help color` command to learn more about them. Also, nothing stops you from defining your own palettes. For example, to get a display with 256 levels of gray, all you need to do is create a `cmap` array as follows:

```
cmap=[0:255]'*ones(1,3)/255;
colormap(cmap);
```

6.1.2 Importing images

If you don’t have an image you can perform tests on in MATLAB®, you can always create one based on raw format images (no header) using image processing software. At the same time, you can save the palette, if that is possible. The following function allows you to read and/or create a file that can be used directly by MATLAB®. The image that was chosen is an image universally used by “image processors” to compare results obtained for different implementations. It is referred to as *lena*. We assume that the data is stored as unsigned 8-bit coded integers.

Figure 6.4 – *Test image*

```
function pixc=raw2matf(NomFE,Nlig,Ncol,Tr,Fc,NomFS)
%%========================================================%
%% Reading a raw image file                               %
%% SYNOPSIS: pixc=RAW2MATF(NomFE,Nlig,Ncol,Tr,Fc,NomFS)   %
%%   NomFE     = raw file ([.raw])                        %
%%   Nlig,Ncol = Image dimensions                         %
%%   Tr        = when ’T’: transposing the image          %
%%   Fc        = when ’F’: creating the file NomFS (.mat) %
%%   NomFS     = Resulting file ([.mat])                  %
%%========================================================%
if nargin<6, NomFS=’fictrav.mat’; end
if nargin<5, Fc=’N’; end
if nargin<4, Tr=’N’; end
```

```
%===== Raw image
nFS=findstr(NomFE,'.');
if isempty(nFS),
   NFE=[NomFE,'.raw'];
else
   NFE=NomFE; NomFE=NomFE(1:nFS-1);
end
fid=fopen(NFE,'r'); [pixc,Npix]=fread(fid,'uchar');
if (Npix ~= Nlig*Ncol)
sprintf('Dimensions error: %d*%d ~= %d',Nlig,Ncol,Npix)
    return
end
pixc=reshape(pixc,Nlig,Ncol); if Tr=='T', pixc=pixc'; end
fclose(fid);
%===== Creating the .MAT file
if Fc=='F',
   sprintf ('Creating the file %s',NomFS)
   eval(['save ' NomFS ' pixc'])
end
return
```

The image can be loaded and displayed (Figure 6.4) by the following program:

```
%===== TSTRAW2MAT.M
pixc=raw2matf('lena50',256,256,'T');
%===== Palette construction
cmap=([255:-1:0]'/255)*[1 1 1];
%===== Displaying with the new palette
imagesc(pixc); colormap(cmap); axis('image')
```

In this program, the palette is defined, but it can also be saved in the image processing application and stored in the `.mat` file.

Comments:

- Recent versions of MATLAB® allow you to directly load and save images in formats such as "bmp" (*bit map*), "tiff", "jpeg", "pcx", etc. using the `imread` and `imwrite` functions.

- Notice that when a palette is used, the `image` function works with an array of integer values (the non-integer values are rounded) between 1 and M. The values above M are constrained to M, and those below 1 are constrained to 1. Type at the end of the previous program:

  ```
  p256=pixc+256; subplot(121); image(p256); axis('image')
  p0=pixc-256; subplot(122); image(p0); axis('image')
  colormap(cmap)
  ```

  You should see a white square and a black square.

– It is usually preferable to use the imagesc function (suffix *sc* as in *scale*) which displays a version with the same scale as the original image: the values are changed to fit between 1 and size(colormap,1).

– The image's color levels can have values such that it becomes difficult to display the image because of a few extreme values. The use of image or imagesc may not be satisfactory. The following function allows you to improve the display by modifying the color distribution:

```
function mydisp(pixr,cmap,stdpar,style)
%%=============================================%
%% Displaying with gray level control          %
%% SYNOPSIS: MYDISP(pixr, cmap, stdpar, style)  %
%%    pixr   = image                            %
%%    cmap   = palette                          %
%%    stdpar = controls the min and max indices %
%%    style  = see AXIS function                %
%%=============================================%
if nargin<2,
    sprintf('Error on arguments');
    return
end
if nargin<4, style='image'; end
if nargin<3, stdpar=3; end
if (stdpar <= 0 | stdpar >10), stdpar=1; end
moy=mean(mean(pixr)); stdp=stdpar*std(std(pixr));
minp=moy-stdp; maxp=moy+stdp;
idx=1+(pixr-minp)*(size(cmap,1)-1)/(maxp-minp);
colormap(cmap); image(idx); axis(style)
return
```

– When using *scanners* or digital cameras, the standard sampling values, in "dots per inch" (*dpi*), are (300 × 600)[1], (600 × 1,200), (1,600 × 3,200), (2,700 × 2,700)..., and for quantification, 8, 10, 12... bits for each of the primary colors.

6.1.3 Arithmetical and logical operations

Because images in MATLAB® are matrices, the usual operations can be directly applied to them. In particular, arithmetic and logical operations between images, pixel by pixel, can be performed from the array values they are associated with.

Thus, the sum of two images pix1 and pix2 of the same size can be written pix1 + pix2, or just as the square root of pix can be written sqrt(pix). You

[1] Meaning 300 dots per inch in one direction, and 600 dots per inch in the other.

only have to make sure that the obtained values are consistent with the color palette, or you can use the functions `imagesc` or `mydisp`.

As for the logical operations applied to the 8 bits of the image pixels' binary representation, the problem is trickier, because MATLAB® has no integer type to which we could directly apply the boolean operations (this was however modified in the recent versions). The operations have to be performed by extracting bits one by one from the image matrices. Here is an example: consider the two images in Figure 6.5 – we are going to perform the AND function between the figure on the left and the figure on the right.

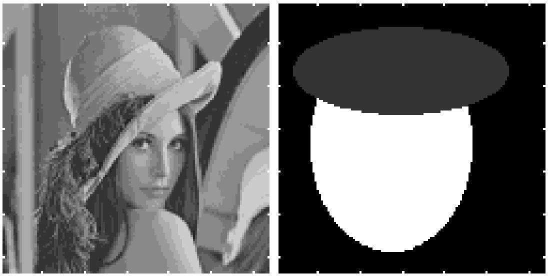


Figure 6.5 – *Logical operation AND*

The result is shown in Figure 6.6: the black areas of the image on the right in Figure 6.5, which are encoded as byte 0000 0000, force the corresponding areas of the resulting image to be black. This is because if `xxxx xxxx` is the value associated with a pixel from the first image, the logical AND of `xxxx xxxx` and `0000 0000` is `0000 0000`. The white areas of the image on the right are encoded as byte 11111111, leaving untouched the values of the corresponding pixels of *Lena*. This is because the logical AND of `xxxxxxxx` with `1111 1111` is `xxxx xxxx`. Finally, the areas of the image on the right which are encoded as `yyyy yyyy` lead to a pixel value with some bits unchanged, and others set to 0.

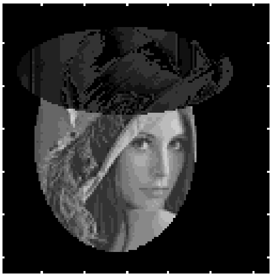


Figure 6.6 – *Result of the logical AND*

The `ANDlog` function performs the logical AND operation we have just described:

```
function pixr=ANDlog(pix1,pix2,L)
%%===========================================%
%% Logical AND between two images            %
%% SYNOPSIS: pixr=ANDLOG(pix1,pix2,L)        %
%%    pix1 = first image (gray palette)      %
%%    pix2 = second image (gray palette)     %
%%    L    = number of bits for color coding %
%%    pixr = image result                    %
%%===========================================%
if (nargin<3), L=8; end
N1=size(pix1);
if (N1 ~= size(pix2)),
    error('Matrix dimensions are not appropriate')
end
pixr=zeros(N1);
%===== Extraction of the bits one by one
for k=1:L
    pixr=pixr+ (rem(pix1,2) & rem(pix2,2)) * 2^(k-1);
    pix1 = fix(pix1/2); pix2 = fix(pix2/2);
end
return
```

The program `testlogic.m` which uses the `ANDlog fonction`, leads to Figure 6.6:

```
%===== TESTLOGIC.M
load lena25;                    % Loading and displaying
subplot(131); imagesc(pixc+1);  % the first image
colormap(cmap); axis('image');
load testlog1;                  % Loading and displaying
subplot(132); imagesc(pixtl+1); % the second image
axis('image')
%===== Logical operation
pixr = ANDlog(pixc,pixtl,size(cmap,1));
subplot(133); imagesc(pixr+1); axis('image');
```

Exercise 6.1 (Logical functions)

1. Write a function that uses the four basic logical operators AND, OR, EOR and NOT, as well as the comparison operators. Use the `eval` function to implement it.

2. Write a test program for the logical operator NOT, as well as for the logical operator that is true when $a_m \leq b_m$, where a_m and b_m are the bits corresponding to two bytes we wish to compare.

6.2 Geometric transformations of an image

6.2.1 The typical transformations

The simple geometric transformations, such as translations, rotations and torsions are problematic because of the "integer" nature of the pixels' position in an image (Figure 6.7).

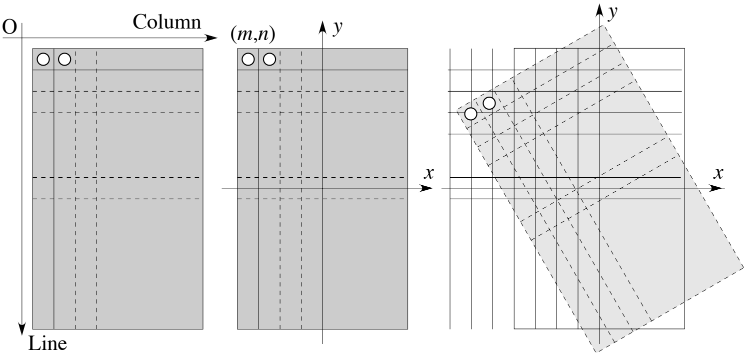


Figure 6.7 – *Rotations of an image*

Example 6.2 (Rotation of an image)
We wish to rotate an image. To make things simpler, we will be using an image in levels of gray.

1. The rotation matrix has the expression:

$$\mathbf{M} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

2. We create an array for the pixel coordinates (change from the line and column numbers over to the x, y coordinates). Applying the rotation to every point provides, after rounding the resulting value, and changing back to the line, column representation, leads us to the final image:

```
%===== GEOMTRANSF.M
% Geometric transformations / Rotation
fmt='jpeg'; fn='imageGG.jpg'; pixc=imread(fn,fmt);
figure(1); imagesc(pixc); Spix=size(pixc);
Nl=Spix(1); Nc=Spix(2);
tbcolor=[0:1/255:1]'*[1 1 1]; % Gray colormap
colormap(tbcolor); set(gca,'DataAspectRatio',[1 1 1])
%===== Rotation center
```

```
xor=(1+Nc)/2; yor=-(1+Nl)/2;
%===== tbidx=indices (columnwise)
tbx=ones(Nl,1)*[1:Nc]; tby=[1:Nl]'*ones(1,Nc);
tbidx=[reshape(tby,1,Nl*Nc);reshape(tbx,1,Nl*Nc)];
idtb=tbidx(1,:)+(tbidx(2,:)-1)*Nl; % Linear indices
%===== tbcoord=coordinates pixels/rotation center
tbcoord=[tbidx(2,:)-xor;-tbidx(1,:)-yor];
%===== Rotation
theta=25; thet=theta*pi/180;
MRot=[cos(thet) -sin(thet);sin(thet) cos(thet)];
tbv=round(MRot*tbcoord);
xmin=min(tbv(1,:)); xmax=max(tbv(1,:)); ncol=xmax-xmin+1;
ymin=min(tbv(2,:)); ymax=max(tbv(2,:)); nlig=ymax-ymin+1;
%===== Index Reconstitution
tbidxR=[-tbv(2,:)-ymin+1;tbv(1,:)-xmin+1];
pixcR=zeros(nlig,ncol); pixcR2=pixcR-1;
idtbR=tbidxR(1,:)+(tbidxR(2,:)-1)*nlig;
pixcR(idtbR)=pixc(idtb); pixcR2(idtbR)=pixc(idtb);
save pixcR2 pixcR2 thet tbcolor Nl Nc; % for next processing
%===== Displaying the result
figure(2); imagesc(pixcR); colormap(tbcolor)
set(gca,'DataAspectRatio',[1 1 1]);
%===== Saving the image for median filtering
pxRmn=min(min(pixcR)); pxRmx=max(max(pixcR));
pixcRn=255*(pixcR-pxRmn)/(pxRmx-pxRmn)+1;
imwrite(pixcRn,tbcolor,'imageGGR.bmp','bmp')
```

Notice the use of the **imread** and **imwrite** functions, making it possible to read and save images in a given format, "jpeg" in this example.

Rotating two neighboring pixels can result, after rounding, in identical coordinates. This leads us to the conclusion that there are "holes" in the target image. These are clearly visible in the image resulting from the rotation (Figure 6.8). We will see in exercise 6.12 how to deal with these isolated points. It is also possible to process the pixels with identical coordinates using a weighted mean of the source pixels.

Generally speaking, *affine* transformations are represented with expression 6.1:

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{6.1}$$

x and y are the coordinates of the source $\mathcal{S}$, X and Y those of the target image $\mathcal{C}$. t_x and t_y define the translation applied to the image.

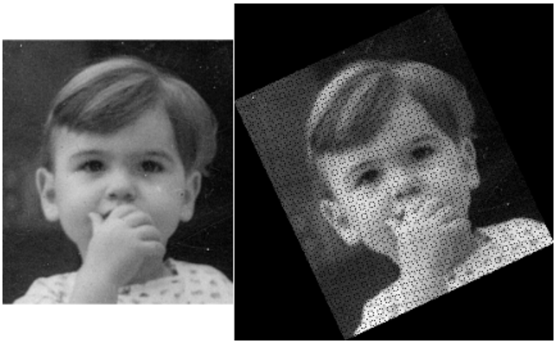


Figure 6.8 – *Flaws due to the rotation*

Likewise, the word *torsion* (see exercise 6.3) is used when the relation between $\mathcal{S}$ and $\mathcal{C}$ is of the type 6.2:

$$\begin{bmatrix} U \\ V \\ T \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ e & f & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ and } X = \frac{U}{T}, Y = \frac{V}{T} \tag{6.2}$$

These two types of transformations pose a problem for interpolation (paragraph 6.5.2) and/or undersampling (paragraph 6.5.1) which we will discuss later.

There is no rule that says you have to use an $x\mathrm{O}y$ axis system instead of a "line, column" coordinate system (LC coordinates). In the case of a rotation, it allows the transformation matrix to preserve its usual form. In general, the transformation matrix can be identified immediately in LC coordinates.

Exercise 6.2 (Plane transformation)

Describing a transformation can be done in an interactive way using a simple shape. Here we are going to use the triangle to define the affine transformation we will apply to the image.

1. Write a linear transformation function of an $n \times m$ pixel image, knowing that the (2×2) transformation matrix is described in an $x\mathrm{O}y$ system.

2. Write a program asking the user to define two triangles interactively, which then calculates the (3×3) affine transformation matrix used to go from one triangle to the other.

3. Apply the transformation to an image.

Exercise 6.3 (Transformation of a rectangular selection)
Many image processing applications allow you to deform a rectangular-shaped selection by having an effect on each corner of the selection. Consider expression 6.2 of the "torsion". For a corner with the coordinates x_k, y_k, the coordinates X_k and Y_k after modifications can be expressed:

$$X_k = \frac{U_k}{T_k} = \frac{ax_k + by_k + t_x}{ex_k + fy_k + 1}$$
$$\text{and } Y_k = \frac{V_k}{T_k} = \frac{cx_k + dy_k + t_y}{ex_k + fy_k + 1}$$

$$\begin{cases} X_k(ex_k + fy_k + 1) = ax_k + by_k + t_x \\ Y_k(ex_k + fy_k + 1) = cx_k + dy_k + t_y \end{cases} \tag{6.3}$$

If applied to all four corners, these expressions make it possible to determine the eight coefficients of the transformation matrix.

1. Using 6.3, determine the linear system needed to find the transformation matrix.

2. Apply this transformation to an image by assuming that the rectangular selection is applied to the whole image.

6.2.2 Aligning images

Many applications – biometrics, identification number recognition, handwriting recognition, etc. – require that an image be forced to fit a certain size before undergoing whatever processing is needed. A method called the *Procrustes method* is often used to perform this operation.

The idea is to start with a simplifed model based on characteristics points. Thus, for a face, we can choose a model such as the one illustrated in Figure 6.9. In the case of a hand, you can either choose points on the outline of the hand, or points on the outline of each finger. For a license plate, the natural choice would be the four corners, etc.

Once we have a reference model $\mathcal{A}$ (the pattern on the left in Figure 6.10), we can start searching for a transformation that drags the characteristic points of the figure $\mathcal{B}$ to be analyzed (the pattern on the right in Figure 6.10) over onto the points of the reference model, according to a criterion used to evaluate the distance between two sets of points.

The fact that the two sets of points $\mathcal{A}$ and $\mathcal{B}$ must correspond exactly adds a difficulty. When the points are provided by the automatic image analysis, $\mathcal{A}$ and $\mathcal{B}$ do not necessarily have the same number of points, meaning that some manual corrections may turn out to be unavoidable.

Let $\mathbf{A} \in \mathbb{R}^{r \times s}$ and $\mathbf{B} \in \mathbb{R}^{r \times s}$ be two $r \times s$ matrices. In our case, the matrix size is $(N, 2)$ or $(2, N)$, where N is the number of charateristic points. We are

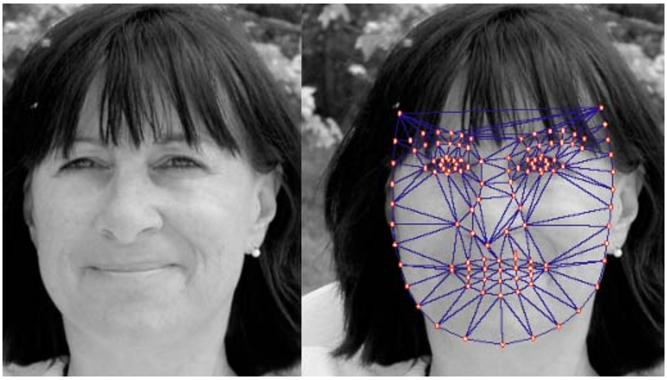


Figure 6.9 – *Triangle-based model*

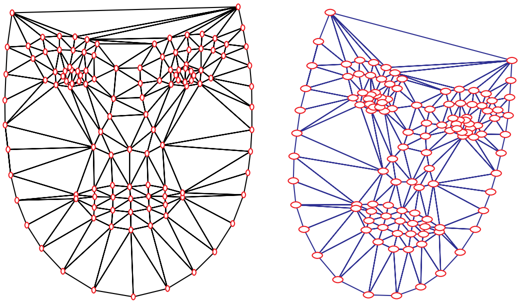


Figure 6.10 – *Characteristic points and Delaunay triangulation: the pattern on the left serves as a reference, the set of points on the right corresponds to the figure we wish to align*

trying to determine the $(r \times r)$ matrix $\mathbf{Q}$, solution to the problem, for which we define a constraint:

$$\begin{cases} \min_{\mathbf{Q}} \|\mathbf{A} - \mathbf{Q}\mathbf{B}\|_F \\ \mathbf{Q}^T\mathbf{Q} = \sigma^2 \mathbf{I}_r \end{cases} \tag{6.4}$$

The matrices $\mathbf{A}$ and $\mathbf{B}$ are centered. If their size is $(N, 2)$:

$$\mathbf{M} = \begin{bmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ \vdots & \vdots \\ X_N & Y_N \end{bmatrix}$$

they are centered by typing `M - ones(N,1)*mean(M)`.

For a matrix $\mathbf{M}$, the Frobenius norm is defined by $||\mathbf{M}||_F^2 = \text{Tr}\{\mathbf{M}\mathbf{M}^T\}$. We have:

$$\begin{aligned} ||\mathbf{A} - \mathbf{Q}\mathbf{B}||_F^2 &= \text{Tr}\{\mathbf{A}\mathbf{A}^T\} - 2\text{Tr}\{\mathbf{Q}\mathbf{B}\mathbf{A}^T\} + \sigma^2\text{Tr}\{\mathbf{B}\mathbf{B}^T\} \\ &= \text{Tr}\{\mathbf{A}\mathbf{A}^T\} - 2\sigma\text{Tr}\{\mathbf{P}\mathbf{B}\mathbf{A}^T\} + \sigma^2\text{Tr}\{\mathbf{B}\mathbf{B}^T\} \qquad (6.5) \end{aligned}$$

where $\mathbf{Q} = \sigma\mathbf{P}$ where $\mathbf{P}$ is a unitary matrix. Hence, for a given σ, the minimization problem amounts to the maximization problem of $\text{Tr}\{\mathbf{P}\mathbf{B}\mathbf{A}^T\}$ under the constraint $\mathbf{P}^T\mathbf{P} = \mathbf{I}_s$.

The matrix $\mathbf{B}\mathbf{A}^T$ is an $r \times r$ square matrix. Its singular value decomposition can be written as follows:

$$\mathbf{B}\mathbf{A}^T = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

where $\mathbf{U}$ and $\mathbf{V}$ are unitary. This leads us to:

$$\text{Tr}\{\mathbf{P}\mathbf{B}\mathbf{A}^T\} = \text{Tr}\{\mathbf{P}\mathbf{U}\mathbf{D}\mathbf{V}^T\} = \text{Tr}\{\mathbf{V}^T\mathbf{P}\mathbf{U}\mathbf{D}\}$$

If we assume $\mathbf{Z} = \mathbf{V}^T\mathbf{P}\mathbf{U}$, and because $\mathbf{D}$ is diagonal, we have:

$$\text{Tr}\{\mathbf{P}\mathbf{B}\mathbf{A}^T\} = \text{Tr}\{\mathbf{Z}\mathbf{D}\} = \sum_i z_{ii}d_{ii}$$

where the z_{ii} and d_{ii} are the diagonal terms of $\mathbf{Z}$ and $\mathbf{D}$ respectively. But, because $\mathbf{Z}$ is a unitary matrix, $|z_{ij}| < 1$ for i, j any pair. Indeed using $\mathbf{Z}^T\mathbf{Z} = \mathbf{I}$, for all j we have $\sum_i |z_{ij}|^2 = 1$. This means that, for any matrix $\mathbf{P}$:

$$\text{Tr}\{\mathbf{P}\mathbf{B}\mathbf{A}^T\} \leq \textstyle\sum_i d_{ii}$$

The upper bound $\sum_i d_{ii}$, which is independent of $\mathbf{P}$, can be reached if we let $\mathbf{Z} = \mathbf{V}^T\mathbf{P}\mathbf{U} = \mathbf{I}$, that is to say:

$$\mathbf{P} = \mathbf{V}\mathbf{U}^T \qquad (6.6)$$

which is unitary. Hence (6.6) is the solution we were looking for. To sum up, after starting with $\mathbf{A}$ and $\mathbf{B} \in \mathbb{R}^{r \times s}$, we calculate, one after the other:

1. $\mathbf{C} = \mathbf{B}\mathbf{A}^T$;
2. the singular value decomposition: $\mathbf{C} = \mathbf{U}\mathbf{D}\mathbf{V}^T$;
3. $\mathbf{P} = \mathbf{V}\mathbf{U}^T$;
4. notice that minimizing 6.5 in regard to σ leads to:

$$\sigma = \frac{\text{Tr}\{\mathbf{P}\mathbf{B}\mathbf{A}^T\}}{\text{Tr}\{\mathbf{B}\mathbf{B}^T\}} \Rightarrow \mathbf{Q} = \frac{\text{Tr}\{\mathbf{P}\mathbf{B}\mathbf{A}^T\}}{\text{Tr}\{\mathbf{B}\mathbf{B}^T\}}\mathbf{P}$$

Notice that if $\mathbf{A} = \mathbf{B}$, $\mathbf{Q} = \mathbf{I}$.

The counterpart to the problem posed by expression (6.4) is determining the $r \times r$ unitary matrix $\mathbf{R}$, solution to the problem:

$$\begin{cases} \min_{\mathbf{R}} \|\mathbf{A} - \mathbf{B}\mathbf{R}\|_F \\ \mathbf{R}^T\mathbf{R} = \beta^2 \mathbf{I}_s \end{cases} \tag{6.7}$$

Of course, its solution can be inferred from the previous one if you notice that 6.7 is equivalent to:

$$\begin{cases} \min_{\mathbf{R}} \|\mathbf{A}^T - \mathbf{R}^T\mathbf{B}^{\mathbf{T}}\|_{\mathbf{F}} \\ \mathbf{R}^T\mathbf{R} = \beta^2 \mathbf{I}_s \end{cases} \tag{6.8}$$

the solution of which is $\mathbf{R} = \beta\mathbf{Y}\mathbf{W}^T$ where $\mathbf{B}^T\mathbf{A} = \mathbf{Y}\mathbf{D}'\mathbf{W}^T$.

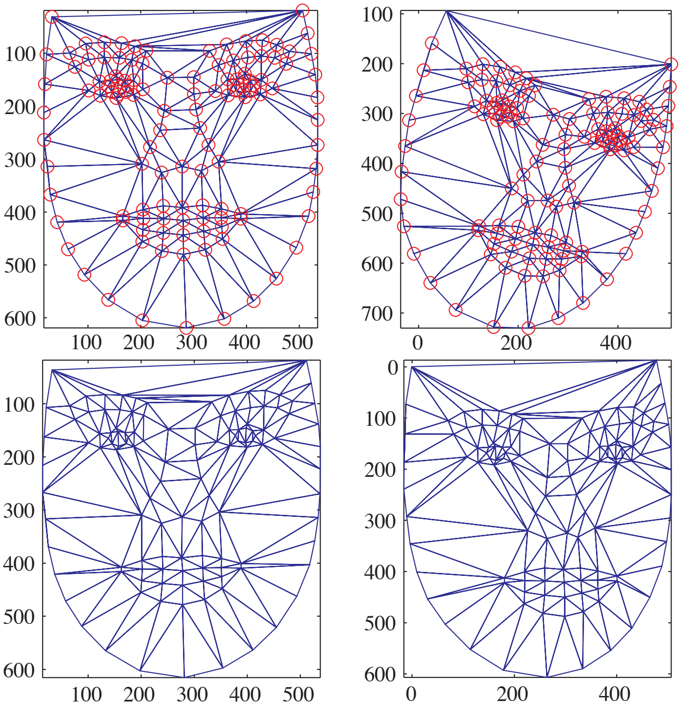


Figure 6.11 – *Applying the multiplications on the right and on the left in the example of Figure 6.10. In the bottom-right, the size of the transformation matrix is* (2×2)*. In the bottom-left, the size of the matrix is* $(N \times N)$ *where* N *is the number of points in the mesh*

Notice in the example above that if one of the dimensions is always equal to 2, for example:

$$\mathbf{A} = \begin{bmatrix} X_1 & X_2 & \dots & X_s \\ Y_1 & Y_2 & \dots & Y_s \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} x_1 & x_2 & \dots & x_s \\ y_1 & y_2 & \dots & y_s \end{bmatrix}$$

the algorithm's development is true for any $r \times s$ pair. In particular, the pixels of two images we wish compare can be directly used.

6.3 Frequential content of an image

Just as it was done for 1D discrete-time signals, we are going to define the Fourier transform $X(\nu, \mu)$ of an image, referred to as the 2D-DTFT.

Definition 6.1 (2D-DTFT) *Let $x(k, \ell)$ be a two index sequence. The 2D-DTFT is the function of ν and of μ defined by:*

$$X(\mu, \nu) = \sum_{k=-\infty}^{+\infty} \sum_{\ell=-\infty}^{+\infty} x(k, \ell) e^{-2\pi j(k\mu + \ell\nu)} \tag{6.9}$$

Because of its definition, $X(\mu, \nu)$is periodic with period 1 for the two variables μ and ν.

In practice, the images processed have a finite size $K \times L$ and we have:

$$X(\mu, \nu) = \sum_{k=0}^{K-1} \sum_{\ell=0}^{L-1} x(k, \ell) e^{-2\pi j(k\mu + \ell\nu)} \tag{6.10}$$

In this case, $X(\mu, \nu)$ poses no existence problems, since the values of $x(k, \ell)$ are bounded and the sequence is finite.

The inverse formula leading to $x(k, \ell)$ from $X(\mu, \nu)$ is:

$$x(k, \ell) = \int_{-1/2}^{1/2} \int_{-1/2}^{1/2} X(\mu, \nu) e^{2\pi j(k\mu + \ell\nu)} d\mu d\nu \tag{6.11}$$

Property 6.1 (2D convolution) *2D convolution is the name of the operation that associates the two sequences $x(k, \ell)$ and $y(k, \ell)$ with the sequence:*

$$z(k, \ell) = (x \star y)(k, \ell) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{\infty} x(i, j) y(k - i, \ell - j) \tag{6.12}$$

The 2D-DTFT of the 2D convolution of x with y is the product of the respective 2-DTFTs. In other words:

$$(x \star y) \leftrightarrow X(\mu, \nu) \times Y(\mu, \nu) \tag{6.13}$$

Just as for 1D, the problem of the numerical calculation of the 2D-DTFT leads to the introduction of the 2D-DFT, which corresponds to the 2D-DTFT's expression calculated in points *regularly spread-out over the* $(0,1)\times(0,1)$ *block.* Without being at all specific, and as for 1D, the number of points before and after the transformation can be considered the same, by completing with zeros if necessary. This leads to the following definition.

Definition 6.2 (2D Discrete Fourier Transform (2D-DFT))
The 2D discrete Fourier transform, or 2D-DFT, of the finite sequence $\{x(k,\ell)\}$*, with* $k \in \{0, \ldots, M-1\}$ *and* $l \in \{0, \ldots, N-1\}$*, is the sequence defined, for* $m \in \{0, \ldots, M-1\}$ *and* $n \in \{0, \ldots, N-1\}$*, by:*

$$X(m,n) = \sum_{k=0}^{M-1}\sum_{\ell=0}^{N-1} x(k,\ell)\exp\left\{-2\pi j\left(\frac{km}{M}+\frac{\ell n}{N}\right)\right\} \tag{6.14}$$

If we change the expression of $X(m,n)$ to:

$$X(m,n) = \sum_{\ell=0}^{N-1}\left(\exp\left\{-2\pi j\frac{\ell n}{N}\right\}\sum_{k=0}^{M-1} x(k,\ell)\exp\left\{-2\pi j\frac{km}{M}\right\}\right) \tag{6.15}$$

for each value of ℓ, the 1D-DFT of the sequence $x(k,\ell)$ for the variable k appears in the parenthesis. With MATLAB®, the N FFTs corresponding to expression 6.15 are calculated by applying the `fft` function to the $(M \times N)$ array `x`. The 2D-DFT is then achieved simply by performing another FFT on the resulting *transpose* array.

To sum up, the 2D-DFT is obtained by doing:

```
fft(fft(x).').'
```

The `fft2` function, available in the basic version of MATLAB®, performs the same operation. As was the case with 1D signals, typing `fft2(x,M,N)` completes, if necessary, the array `x` with zeros so as to have an $M \times N$ array. Again, as it was the case for 1D signals, it is often preferable to display the spatial frequencies with values between 0 and 1, or between 0 and 1/2. This is what we did in example 6.3.

Example 6.3 (2D-DTFT of a square block)
The following program calculates the 2D-DTFT of a square block and displays its modulus. The result is shown in Figures 6.12 and 6.13:

```
%===== TSTFFTBLOCK.M
block=zeros(8,8); delta=4;
block(1:delta,1:delta)=ones(delta,delta);
set(gcf,'color',[1 1 1])
subplot(131); imagesc(block); colormap('gray');
```

```
axis('image'); set(gca,'xcolor',[0 0 0],'ycolor',[0 0 0])
%===== Spectral content
M=32; N=32; blockFqs=fft2(block,M,N);
%===== Normalized spatial frequencies
mu=(0:M-1)/M;nu=(0:N-1)/N;
subplot(132); contour(nu,mu,abs(blockFqs),20);
axis('square'); set(gca,'xlim',[0 .5],'ylim',[0 .5])
set(gca,'xcolor',[0 0 0],'ycolor',[0 0 0])
subplot(133); imagesc(nu,mu,abs(blockFqs))
axis('square'); set(gca,'xlim',[0 .5],'ylim',[0 .5])
set(gca,'xcolor',[0 0 0],'ycolor',[0 0 0])
```

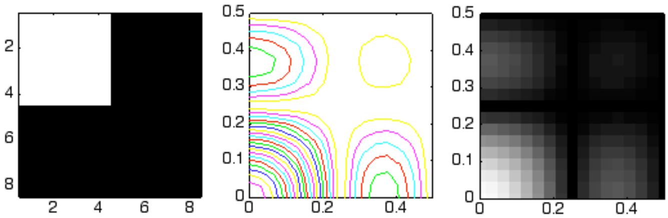


Figure 6.12 – *2D-FFT applied to the rectangular block by restricting the frequencies to* $([0, 1/2] \times [0, 1/2])$

The lobes are similar to the ones obtained for the discrete-time sine cardinal (Figure 6.13).

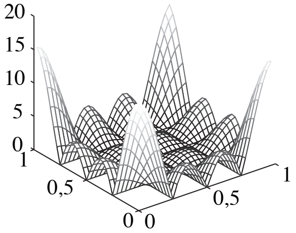


Figure 6.13 – *2D-FFT applied to the rectangular block with the frequencies belonging to* $([0, 1] \times [0, 1])$

The properties of the 2D-DFT are similar to those of the 1D-DFT:

Property 6.2 (Inverse 2D-DFT) *The 2D-inverse-DFT of $X(m, n)$ has the expression:*

$$x(k, \ell) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X(m, n) \exp\left\{2\pi j \left(\frac{km}{M} + \frac{\ell n}{N}\right)\right\}$$

where $k \in \{0, \ldots, M-1\}$ *and* $\ell \in \{0, \ldots, N-1\}$,

This result is obtained by using the relation:

$$\begin{aligned} g(k,\ell) &= \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \exp\left\{2\pi j \left(\frac{km}{M} + \frac{\ell n}{N}\right)\right\} \\ &= \begin{cases} 1 & \text{if } k = 0 \bmod M \text{ and } \ell = 0 \bmod N \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Property 6.3 (Circular convolution (2D-DFT))
Let $x(k,\ell)$ *and* $y(k,\ell)$ *be two images with the same finite size* $M \times N$*. Let* $X(m,n)$ *and* $Y(m,n)$ *be their respective 2D-DFTs calculated over* $M \times N$ *points. Then the inverse 2D-DFT of* $Z(m,n) = X(m,n)Y(m,n)$ *has the following expression, for* $k \in \{0, \ldots, M-1\}$ *and* $\ell \in \{0, \ldots, N-1\}$*:*

$$z(k,\ell) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} x(u,v) y((k-u) \bmod M, (\ell - v) \bmod N)$$

where the indices of y *are calculated modulo* M *and modulo* N *respectively.*

Property 6.4 (Real image and hermitian symmetry (2D-DFT))
If the image $x(k,\ell)$ *is real then its 2D-DFT is such that:*

$$X(m,n) = X^*(-m \bmod M, -n \bmod N)$$

where the first indices[2] *are calculated modulo* M *and the second indices modulo* N*.*

Thus, $X(0,0) = X^*(0,0)$ which is therefore real. If $M = 8$ and $N = 16$, $X(4,3) = X^*(8-4, 16-3) = X^*(4,13)$.

Example 6.4 (2D-DFT of a checkerboard)
The following program calculates the 2D-DFT of a checkerboard the horizontal frequency of which is `f0x=0.2` and the vertical frequency `f0y=0.3`, and displays its modulus. The resulting graph shows lobes at the spatial frequencies $(0.2; 0.3)$ and $(1-0.2; 1-0.3)$, since the image is real. You can try other values of `f0x` and `f0y`.

```
%===== TSTFFTMO.M
%===== Checkerboard
clear; cote=8; bloc=zeros(cote,cote);
f0x=0.2; f0y=0.3;
dom=f0x*(0:cote-1)'*ones(1,cote)+f0y*ones(cote,1)*(0:cote-1);
chkbd=cos(2*pi*dom)+1;
```

[2] Bear in mind that the array indices start at 1 and not 0.

```
set(gcf,'color',[1 1 1])
subplot(121); imagesc(chkbd);
colormap('gray'); axis('image')
set(gca,'xcolor',[0 0 0],'ycolor',[0 0 0])
%===== Spectral content
M=128; N=128; chkbdFqs=fft2(chkbd,M,N);
mu=(0:M-1)/M;nu=(0:N-1)/N;
subplot(122);
contour(nu,mu,abs(chkbdFqs),20); %imagesc(nu,mu,abs(chkbdFqs));
set(gca,'xcolor',[0 0 0],'ycolor',[0 0 0])
axis('square'); grid
```

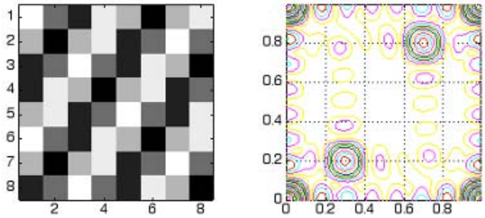


Figure 6.14 – *2D-FFT applied to a checkerboard*

If $x(k,\ell)$ is separable, that is if $x(k,\ell) = x_1(k)x_2(\ell)$, the 2D-DFT can be expressed as the product of two 1D-DFTs (meaning that $X(m,n)$ is separable). Thus, we can write:

$$\begin{aligned} X(m,n) &= \sum_{k=0}^{M-1} x_1(k)\exp\left\{-2\pi j\frac{km}{M}\right\} \times \sum_{\ell=0}^{N-1} x_2(\ell)\exp\left\{-2\pi j\frac{ln}{N}\right\} \\ &= X_1(m) \times X_2(n) \end{aligned}$$

The calculation then becomes quite simpler.

6.4 Linear filtering

The `filter2` function, used for 2D filtering, is available in the basic version of MATLAB®. This function uses the 2D-convolution function, the command line of which is, in MATLAB®, `c=conv2(a,b)`. The 2D-convolution is a *built-in function*.

Definition 6.3 *2D-linear filtering is the operation that associates the image $x(k,\ell)$ with the image $y(k,\ell)$ defined by:*

$$y(k,\ell) = (x \star h)(k,\ell) = \sum_{m=-\infty}^{+\infty}\sum_{n=-\infty}^{+\infty} x(k-m,\ell-n)h(m,n) \qquad (6.16)$$

The two index sequence $h(k,\ell)$, characteristic of the filter, is called the Point Spread Function, *or* PSF.

As was the case for 1D-filtering, the operation denoted by "$\star$" in expression 6.16 is linear and *space-invariant*, and the sequence $h(k,\ell)$ is the equivalent of the impulse response for the one dimension case. Once again, property 6.1 leads to a simple expression of the filtering operation in the frequential range. This gives us the following property.

Property 6.5 *Consider a 2D-linear filter with the PSF $h(k,\ell)$. $H(\mu,\nu)$ denotes the 2D-DTFT of its PSF. It is called the* optical transfer function, *or* OTF. *Because of property 6.1 we have:*

$$Y(\mu,\nu) = H(\mu,\nu)X(\mu,\nu)$$

where $X(\mu,\nu)$ and $Y(\mu,\nu)$ refer to the 2-DTFTs of $x(k,\ell)$ and $y(k,\ell)$ respectively.

Thus, the *identity filter* has the PSF $h(k,\ell) = \delta(k)\delta(\ell)$, where $\delta(k)$ is equal to 1 if $k = 0$ and 0 otherwise. Its OTF is equal to 1 for any frequency pair (μ,ν). This filter leaves the input image untouched. Because of property 6.5, we can also say that the identity filter passes all frequencies.

In MATLAB®, unlike the `filter(b,a,x)` function for 1D use, which allows the user to design an infinite impulse response filter using the input coefficients `a`, the `filter2(B,x)` function performs only the 2D equivalent of a finite impulse response filtering, the expression of which is:

$$y(k,\ell) = (x \star h)(k,\ell) = \sum_{m=M_1}^{M_2} \sum_{n=N_1}^{N_2} x(k-m,\ell-n)h(m,n) \tag{6.17}$$

The `filter2` function has an additional parameter that allows the user to set how the side effects should be taken into account: `'same'` to have an output image with the same size as the input image (this is the default option), `'valid'` to keep only the part of the image unaffected by the side effect (the resulting image is smaller than the original), and `full` to keep all of the points, including the ones resulting from the filter's impulse response (this leads to an image larger than the original).

The concept of *stability* is essential, as it was with 1D signals. It states that to any bounded input corresponds a bounded output. Because we will only be considering filters characterized by expression 6.17 and similar to the 1D FIR filters, the stability condition will always be met from now on.

On the other hand, the concept of *causality*, although fundamental when it comes to signals, has very little significance in the case of images. This is because there is no reason for the quantity calculated for the coordinates

(k, ℓ) to be dependent only on the points placed "before" (k, ℓ), that is to say $(k - m, \ell - n)$, where m and n are positive. In 2D processing, all of the points around (k, ℓ) can contribute to the calculated value.

Example 6.5 (Circular filter)
Consider what is called the *circular filter*, $h(k, l)$, defined in the program:

```
%===== SMOOTH1.M
h =[0 0 1 1 1 0 0;
    0 1 1 1 1 1 0;
    1 1 1 1 1 1 1;
    1 1 1 1 1 1 1;
    1 1 1 1 1 1 1;
    0 1 1 1 1 1 0;
    0 0 1 1 1 0 0];
h = h / sum(sum(h));
load wenmanu; subplot(121); imagesc(pixc);
colormap(cmap); axis('image');
set(gca,'units','pixels','DataAspectRatio',[1 1 1])
pixr=filter2(h,pixc); subplot(122); imagesc(pixr);
set(gca,'units','pixels','DataAspectRatio',[1 1 1])
axis('image')
```

This program smooths the image.

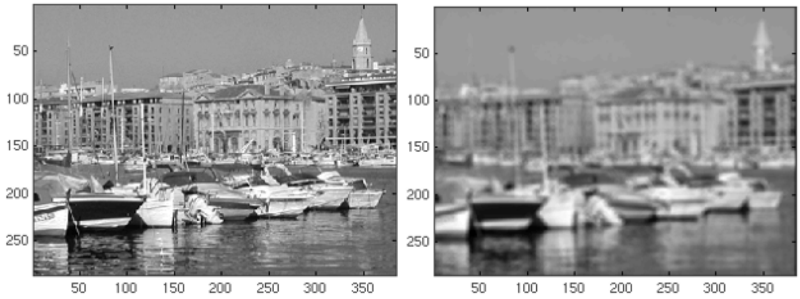


Figure 6.15 – *Smoothing of an image test using the circular filter*

As was the case with 1D smoothing filters, this filter tends to "erase" high frequencies, particularly the ones contained in the contours, and therefore produces a blurred image (Figure 6.15) compared to the original image.

Definition 6.4 *A filter is said to be separable when its PSF is such that:*

$$h(k, \ell) = h_x(k) h_y(\ell) \tag{6.18}$$

In the case of a finite PSF, if $\mathbf{h}$ *is the matrix with* $h(k,\ell)$ *as its elements, and if* $\mathbf{h}_x$ *and* $\mathbf{h}_y$ *are the vectors with the respective components* $h_x(k)$ *and* $h_y(\ell)$*, relation 6.18 is equivalent to:*

$$\mathbf{h} = \mathbf{h}_x \mathbf{h}_y^T \tag{6.19}$$

We are going to show that a separable 2D filtering can be performed by combining two consecutive 1D filters. This is how it works:

$$\begin{aligned}
(x \star h)(m,n) &= \sum_{k=K_1}^{K_2} \sum_{\ell=L_1}^{L_2} x(m-k, n-\ell) h(k,\ell) \\
&= \sum_{k=K_1}^{K_2} h_x(k) \left(\sum_{\ell=L_1}^{L_2} x(m-k, n-\ell) h_y(\ell) \right) \\
&= \sum_{\ell=L_1}^{L_2} h_y(\ell) \left(\sum_{k=K_1}^{K_2} x(m-k, n-\ell) h_x(k) \right)
\end{aligned}$$

Bear in mind that if the `filter` function is used for a separable 2D filtering, you must take into account the fact that `filter` implements a causal design. (exercise 6.4).

Exercise 6.4 (The rectangular filter)
Consider the rectangular filter defined by:

$$\mathbf{h} = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \frac{1}{25} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} [1\ 1\ 1\ 1\ 1]$$

Write a program that:

1. Performs the filtering of the test image;
2. Performs the same filtering using two separate 1D filterings.

Exercise 6.5 (The conical filter)
The conical filter is defined by:

$$\mathbf{h} = \frac{1}{25} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 2 & 2 & 0 \\ 1 & 2 & 5 & 2 & 1 \\ 0 & 2 & 2 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Apply the conical filter to the test image.

Definition 6.5 (The Gaussian smoothing filter) *The generating element of the Gaussian smoothing filter's PSF is:*

$$h(k,\ell) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{k^2+\ell^2}{2\sigma^2}\right)$$

This filter is separable, since we can write $h(k,\ell) = h_x(k)h_y(\ell)$.

The smaller the σ parameter is, the more the filter behaves like an identity filter, that is to say that it passes all the frequencies of the plane. The gain filter can of course be modified by mulitplying it by a constant.

Exercise 6.6 (The Gaussian smoothing filter)

1. Write a MATLAB® function that calculates the PSF of Gaussian smoothing filter using σ.

2. Apply the Gaussian filter to the test image.

2D-DFT frequency filtering

Starting with the circular convolution property 6.3, it is possible to consider performing a filtering by simply multiplying the 2D-DFT of an image by the 2D-DFT of the filter's PSF. Of course, just like in 1D, this process must take into account the circular convolution property.

Consider the PSF $h(k,\ell)$ of a $K \times L$ filter (number of non-zero coefficients), and an $M \times N$ image $x(k,\ell)$. We will assume $M > K$ and $N > L$. $H(m,n)$ and $X(m,n)$ refer to the 2D-DFTs of the PSF and of the image respectively. Both are calculated for $M \times N$ points. According to property 6.3, the inverse 2D-DFT of the product $H(m,n)X(m,n)$ can be written, for $k \in \{0,\dots,M-1\}$ and $\ell \in \{0,\dots,N-1\}$:

$$y(k,\ell) = \sum_{u=0}^{K-1}\sum_{v=0}^{L-1} h(u,v)x(k-u \bmod M, \ell - v \bmod N)$$

For $k \geq K-1$, $(k-u \bmod M) = k-u$: there is no index "aliasing" when we sum u from 0 to $(K-1)$. This also true for $\ell \geq L-1$, $(\ell - v \bmod N) = \ell - v$. In this case, the calculated points do correspond to those of the convolution associated with the filtering. However, for $k < K-1$ and/or $\ell < N-1$, there is an index "aliasing" which leads to an incorrect result. One way of avoiding this phenomenon is by completing the image with K zeros along the horizontal axis, and L zeros along the vertical axis.

Derivative operations

To display the variations of a 2D function, the concept of derivative can be used, as it was in 1D. The difference is that in 2D, the derivative comprises two components corresponding to the two directions of the plane. Thus, to perform a 2D-derivative, you can use a first derivative filter along the horizontal direction, and a second one along the vertical direction.

Definition 6.6 (Prewitt derivative filter)
The PSF of a Prewitt derivative filter along the vertical direction is given by:

$$\mathbf{h}_v = \frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

and the PSF of a Prewitt derivative filter along the vertical direction by:

$$\mathbf{h}_h = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

These two filters are therefore separable. Decomposing $\mathbf{h}_v$ clearly shows:

- a smoothing function along one direction, smoothing corresponding to the vector $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}/\sqrt{3}$;
- a derivative function in the other direction, corresponding to the vector $\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}/\sqrt{3}$. Remember that the 1D causal filter defined by this vector associates the input $u(n)$ with the output $v(n) = (u(n) - u(n-2))/\sqrt{3}$, which can be seen as the derivative.

Definition 6.7 (Sobel derivative filter)
The PSF of a Sobel fitler along the vertical axis is given by:

$$\mathbf{h}_v = \frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \tag{6.20}$$

The PSF of a Sobel fitler along the vertical axis is given by:

$$\mathbf{h}_h = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \tag{6.21}$$

Sobel filters are separable. Notice that they perform a derivative along one axis, and a smoothing operation along the perpendicular axis.

Applying formula 6.19 leads to a 2D filter that derives in both directions without any smoothing:

$$\mathbf{h} = \alpha \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = \alpha \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

where α is a normalization coefficient.

Starting off in 1D, we can also design a 2D filter that approximates the second derivative, by convoluting the impulse response filter $\begin{bmatrix} -1 & 1 \end{bmatrix}$, which is an approximation of the first derivative, with itself. If you type `conv([-1 1],[-1 1])` in MATLAB®, the result is `[1 -2 1]`. By combining the two directions, we get a 2D filter that performs a second derivative in both directions, defined by:

$$\mathbf{h} = \alpha \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} = \alpha \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{6.22}$$

where α is a normalization coefficient.

Generally speaking, it is of course possible to design other filters using one dimension design methods, and then inferring a 2D separable filter with formula 6.19.

Exercise 6.7 (The Sobel derivative filter)

1. Apply the Sobel filters 6.20 and 6.21 to the test image.
2. Apply the filter 6.22 to the same image.
3. By using a similar method to the window method, calculate a derivative filter.
4. Same question for a second derivative filter.

Definition 6.8 (Gaussian derivative filter)
Consider the function defined as the difference between two Gaussians, also called a Difference of gaussians mask, or DoG mask:

$$g(k,\ell) = \frac{1}{2\pi\sigma_1^2} \exp\left(-\frac{k^2+\ell^2}{2\sigma_1^2}\right) - \frac{1}{2\pi\sigma_2^2} \exp\left(-\frac{k^2+\ell^2}{2\sigma_2^2}\right)$$

with $\sigma_2 = r\sigma_1$ and r between 1.4 et 1.8. The Gaussian derivative filter is the filter with the following PSF:

$$h(k,\ell) = g(k,\ell) - \sum_k \sum_\ell g(k,\ell)$$

implying that $\sum_k \sum_\ell h(k,\ell) = 0$.

The imposed condition, $\sum_k \sum_\ell h(k,\ell) = 0$, is related to the fact that a derivative filter has a gain equal to 0 at the frequency 0. This result is similar to the one obtained in 1D in exercise 4.11.

The graph of a Gaussian derivative filter PSF is shaped like the one in Figure 6.16.

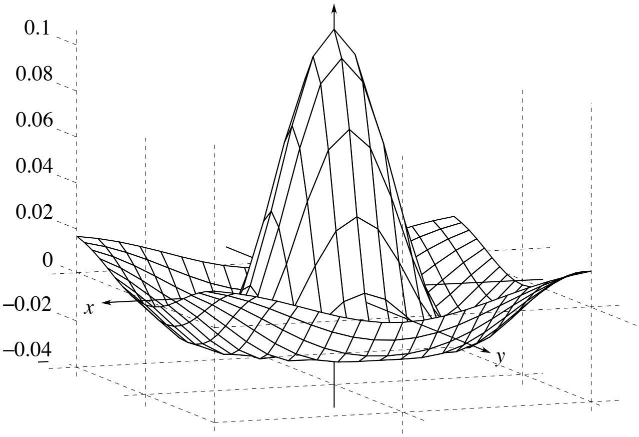


Figure 6.16 – *Graph shape of a Gaussian derivative filter's PSF*

Example 6.6 (Gaussian derivative filter)

1. Write a MATLAB® function that calculates the PSF of a Gaussian derivative filter. Write it so that $\sum_k \sum_l h(k,l) = 0$.
2. Apply this filter to the test image for three values of $\sigma_1 = \{1, 2, 3\}$ and for $r = 1.4$. Save the results in three different files (use the functions `sprintf` and `eval` to change in a program the name of the saved file).

SOLUTION:

1. Type:

```
function hd=dergauss(sigma)
%%===============================%
%% Gaussian derivative filter    %
%% SYNOPSIS: hd = DERGAUSS(sigma) %
%%   sigma = Standard deviation  %
%%   hd    = filter PSF (N*N)    %
%%===============================%
rho=[-sigma*3:sigma*3]; N=length(rho);
rp=1.4; s2=2*sigma^2; s22=s2*rp*rp;
```

```
idx= ([1:N]-(N+1)/2)' * ones(1,N); idy=idx';
idxa=[1:N]' * ones(1,N); idya=idxa';
%=====
indices(1,:)=reshape(idx,1,N*N);
inda(1,:)=reshape(idxa,1,N*N);
indices(2,:)=reshape(idy,1,N*N);
inda(2,:)=reshape(idya,1,N*N);
rho2=sum(indices .* indices); rho=sqrt(rho2);
for k=1:N*N
    g1=(1/sigma)*exp(-rho2(k) / s2);
    g2=(1/sigma/rp)*exp(-rho2(k) / s22);
    hd(inda(2,k),inda(1,k))=g1-g2;
end
hd=hd-sum(sum(hd))/N/N;
return
```

2. Applying the filter to the test image (Figure 6.17):

```
%===== TSTDERGAUSS.M
%===== Loading the image
clear; load lena; subplot(221); imagesc(pixc+1);
colormap(cmap); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
%===== Gaussian derivative filter
for k=1:3
    hd=dergauss(k);
    pixr=round(filter2(hd,pixc));
    subplot(2,2,k+1); imagesc(pixr); axis('image')
    set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
end
set(gcf,'Color',[1 1 1])
```

The programs we have just described are used in the contour detection program.

Definition 6.9 (Gaussian derivative-smoothing filter)
Consider the rotation of angle θ that changes the point with coordinates (u, v) according to the expression:

$$\begin{bmatrix} u(k,\ell) \\ v(k,\ell) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} k \\ \ell \end{bmatrix} \tag{6.23}$$

Consider the Gauss function:

$$h_1(k,\ell) = \frac{1}{\sigma_1\sqrt{2\pi}} \exp\left(-\frac{u(k,\ell)^2}{2\sigma_1^2}\right) \tag{6.24}$$

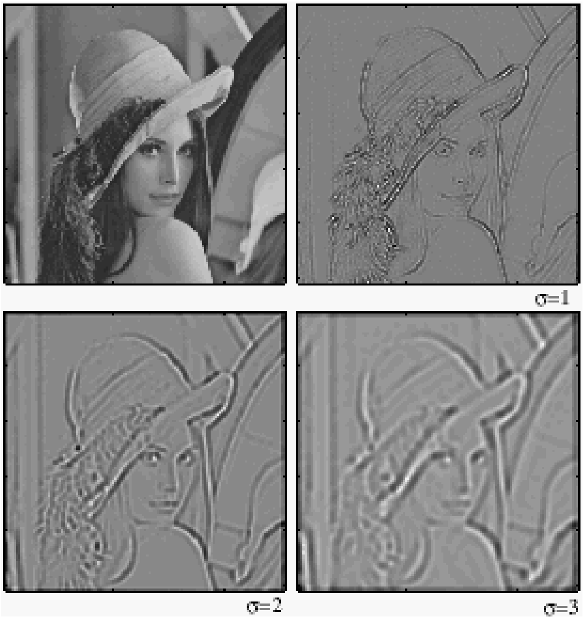


Figure 6.17 – *Gaussian derivative for $\sigma_1 = \{1, 2, 3\}$ and $\sigma_2 = 1.4\sigma_1$*

and the derivative function:

$$h_2(k,\ell) = -\frac{v(k,\ell)}{\sigma_2^3\sqrt{2\pi}} \exp\left(-\frac{v(k,\ell)^2}{2\sigma_2^2}\right) \tag{6.25}$$

The Gaussian derivative-smoothing filter is the filter that performs a Gaussian smoothing filtering (function h_1) along the direction $\theta \in (0, 2\pi)$ and a Gaussian derivative filtering (function h_2) along the perpendicular direction.

The expression of its PSF's generating element is:

$$h(k,\ell) = h_1(k,\ell)h_2(k,\ell) - \sum_k \sum_\ell h_1(k,\ell)h_2(k,\ell)$$

which verifies $\sum_k \sum_\ell h(k,\ell) = 0$.

Exercise 6.8 (Gaussian derivative-smoothing filter)

1. Write a MATLAB® function that calculates the PSF of a Gaussian derivative-smoothing filter.

2. Apply this filter to the test image.

6.5 Other operations on images

6.5.1 Undersampling

As it was the case with one dimension signals, the undersampling has to meet some conditions to avoid the aliasing phenomenon. Remember that aliasing occurs when the sampling rate is too slow compared to the frequencies found in the image, and causes frequential artifacts to appear. In an image, high frequencies correspond to important variations in color and/or brightness concentrated on small surfaces. Take the example of the images represented in Figure 6.18. They were obtained with the following program:

```
%===== ALIASINGTRAINS
close all; clear all
load trainsV4
xx1se=xx1(1:5:512,1:5:768); % Under-sampling
lwpass=ones(5,5)/25;
yy1=filter2(lwpass,xx1);    % Filtering before
yy1se=yy1(1:5:512,1:5:768); % under-sampling
subplot(221); imagesc(xx1); colormap('gray')
subplot(222); imagesc(xx1se); colormap('gray')
subplot(223); imagesc(yy1); colormap('gray')
subplot(224); imagesc(yy1se); colormap('gray')
set(gcf,'Color',[1 1 1])
```

In the top-left corner, you can see the original image, a 512 × 768 array. This image contains “high frequencies”, particularly around the electric cables and the tracks, where the shapes are in some places less than a few pixels wide. In the image represented in the top-right corner, obtained by taking 1 out of 5 pixels horizontally and vertically, you can clearly see major and erratic variations in some areas of the image, due to aliasing.

Just like in 1D, a low-pass filtering must be performed before the undersampling. This can be done with a simple filter, calculating the mean over 5 × 5 cells. This operation is performed by `filter2(lwpass,xx1)`, which uses the `filter2` function. The resulting image is shown in the bottom-left corner. The filter causes a slight “blur”. The image in the bottom-right corner shows the previous image after the undersampling operation. Most of the artifacts are gone. The tracks in particular show less unwanted fluctuations.

6.5.2 Oversampling

As for the oversampling of 1D signals, the interpolation operation can be performed by the insertion of “zeros” (the zero's significance is not the same here) followed by a low-pass filter. In the following example, we isolated the part of the original image containing the clock, on the platform to the left. This portion of the image is shown on the left-hand side of Figure 6.19. In order

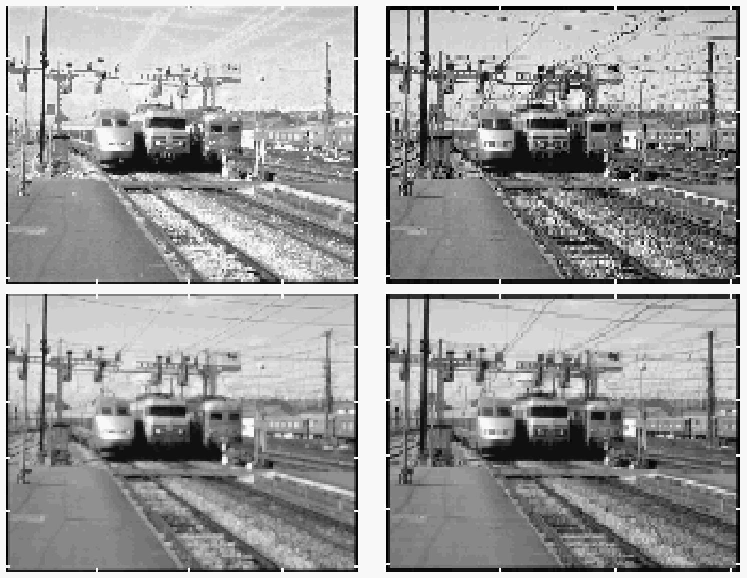


Figure 6.18 – *Effects of spectrum aliasing. In the top-left corner, the original image (*512×768*). In the top-right corner, the same image undersampled by a factor of* 5*. In the bottom-left corner, the image filtered by a smoothing filter over a* 5×5 *square. In the bottom-right corner, the image filtered and undersampled by a factor of* 5

to improve the image rendering, we oversampled by a factor of 4, horizontally and vertically. The low-pass filter is a separable filter with a PSF of the type $\sin\alpha/\alpha$, to which a Hamming window is applied in order to reduce the ripples in the resulting image. The following program was used to obtain the image on the right of Figure 6.19:

```
%===== OVERSAMP2DS.M
%===== Over-sampling ratio
clear; Mx=4; My=4; cmap='gray';
load trainsV4; ima=xx1;       % The file "trains" --> xx1
%===== Zooming in on the clock
pixc=ima(180:210,80:120);
[Lig,Col]=size(pixc);
%===== Low-pass filter PSF (Lfft>N)
N=30; [X,Y]=meshgrid(-N:1:N, -N:1:N); X=X+eps; Y=Y+eps;
FEP=Mx*My*(sin(pi * X/Mx) ./ X) .* (sin(pi * Y/My)./ Y);
%===== Hamming window
W = (0.54 - 0.46*cos(2*pi*(X+N)/(2*N))) ...
    .* (0.54 - 0.46*cos(2*pi*(Y+N)/(2*N)));
FEP=FEP .* W;
%===== Expansion and filtering
```

```
pixcz=zeros(Mx*Lig,My*Col);
pixcz(1:Mx:Mx*Lig,1:My:My*Col)=pixc;
pixcSE=filter2(FEP,pixcz);
%===== Displaying the result
subplot(121); imagesc(pixc); axis('image'); colormap(cmap);
subplot(122); imagesc(pixcSE); axis('image'); colormap(cmap);
set(gcf,'Color',[1 1 1])
```

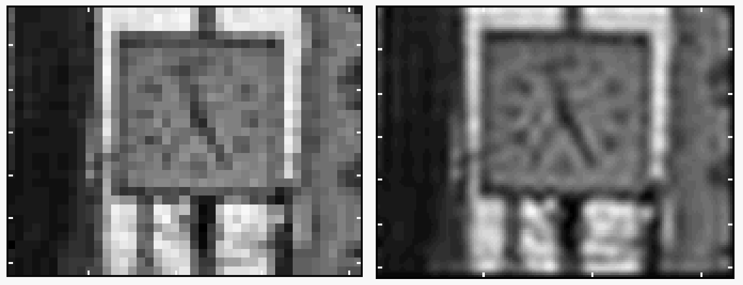


Figure 6.19 – *Image on the left: zoom-in on the clock in the original image from Figure 6.18. Image on the right: oversampling by a factor of* 4 *in both directions*

These techniques, taken directly from signal processing, are not the only ones used. The bibliography shows some sources of information for bilinear interpolations, cubic interpolations, etc.

For example, the bilinear interpolation consists of constructing "intermediate" pixels P from four pixels P_{00}, P_{01}, P_{10} and P_{11} by making the values of the parameters t_x and t_y vary from 0 to 1:

$$P = P_{00}(1-t_x)(1-t_y) + P_{01}(1-t_x)t_y + P_{10}t_x(1-t_y) + P_{11}t_xt_y$$

Example 6.7 (Bilinear interpolation) The `bilintrimg` function performs the bilinear interpolation of an image with $(n \times m)$ pixels:

```
function pixcR=bilintrimg(pixc,Rintx,Rinty)
%%============================================%
%% Bilinear interpolation of an image         %
%% SYNOPSIS: pixcR=BILINTRIMG(pixc,Rintx,Rinty) %
%%    pixc = Image (nl*nc) pixels             %
%%   Rintx = Interpolation rate (x)           %
%%   Rinty = Interpolation rate (y)           %
%%============================================%
Spix=size(pixc); Nl=Spix(1); Nc=Spix(2);
txt=[0:Rintx-1]/Rintx; ty=[0:Rinty-1]'/Rinty;
nlig=(Nl-1)*Rinty+1; ncol=(Nc-1)*Rintx+1;
pixcR=zeros(nlig+Rinty,ncol+Rintx);
```

```
M00=(1-ty)*(1-txt); M01=(1-ty)*txt;
M10=ty*(1-txt); M11=ty*txt;
pixc=[pixc zeros(Nl,1);zeros(1,Nc+1)];
for kl=1:Nl
    for kc=1:Nc
        tl=(kl-1)*Rinty+[1:Rinty]; tc=(kc-1)*Rintx+[1:Rintx];
        PC=pixc(kl,kc)*M00+pixc(kl,kc+1)*M01+...
           pixc(kl+1,kc)*M10+pixc(kl+1,kc+1)*M11;
       pixcR(tl,tc)=PC;
   end
end
pixcR=pixcR(1:nlig,1:ncol);
return
```

The following program loads an image in levels of gray and oversamples by a factor of Rint using a bilinear transformation (Figure 6.20):

```
%===== INTBILIN.M
xor=40; yor=40; % Positionning in the window
pixc1=imread('oeile.jpg','jpeg');
Spix=size(pixc1); Nl=Spix(1); Nc=Spix(2);
pixc=zeros(Spix); pixc(:)=pixc1;
cmap=[0:1/255:1]'*[1 1 1];
subplot(121); imagesc(pixc); colormap(cmap)
set(gca,'units','pixels','Position',[xor yor Nc Nl]);
set(gcf,'color',[1 1 1])
%===== Interpolation with ratio 4 in both directions
Rint=4;
pixcR=bilintrimg(pixc,Rint,Rint);
subplot(122); imagesc(pixcR); colormap(cmap)
Spix=size(pixcR); ncol=Spix(2); nlig=Spix(1);
set(gca,'units','pixels','Position',[xor+xor+Nc yor ncol nlig]);
```

6.5.3 Contour detection

Contour detection is a common application of image processing. A simple method is to start by extracting the portions of the image with a significant gradient. This can be done with a derivative filter. We then need to define a boolean information, for each pixel, stating whether or not the pixel belongs to a contour. This can be done simply by comparing the obtained results to a threshold. The following program uses an image obtained by differentiation in example 6.6:

```
%===== THRESHOLDG.M
%===== File loading
clear; load lenabool2;
subplot(121); mydisp(pixr,cmap);
%===== Threshold with manual choice of alpha
```

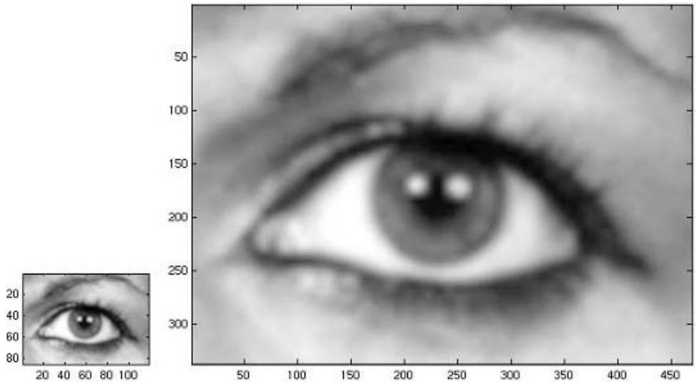


Figure 6.20 – *Applying the bilinear interpolation*

```
alpha=0;
decal=round((max(max(pixr))+min(min(pixr)))/2);
subplot(122); imagesc(-sign(pixr+decal+alpha));
axis('image')
```

The result of the thresholding is represented in Figure 6.21.

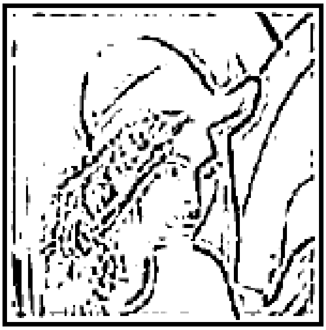


Figure 6.21 – *Results of the thresholding after derivative filtering*

It can be wiser to search for local maxima – peaks – in the result of the derivative. The following function can find the maxima:

```
function resul=searchmax(pixr,cmap)
%%==================================%
%% Searching the peaks              %
%% SYNOPSIS resul=RECHMAX(pixr,cmap) %
%%   pixr  = image                  %
%%   cmap  = palette                %
%%   resul = result                  %
%%===================================%
[nlig, ncol]= size(pixr); Lcmap=size(cmap,1);
%=====
pp=3;
pixr(:,1:pp)=zeros(nlig,pp); pixr(1:pp,:)=zeros(pp,ncol);
pixr(:,ncol-pp+1:ncol)=zeros(nlig,pp);
pixr(nlig-pp+1:nlig,:)=zeros(pp,ncol);
%===== Searching the maxima
pixdy1= [pixr(2:nlig,:); pixr(nlig,:)];
pixdy2= [pixr(1,:); pixr(1:nlig-1,:)];
pixdx1= [pixr(:, 2:ncol) pixr(:,ncol)];
pixdx2= [pixr(:,1) pixr(:,1:ncol-1)];
maxima = find((pixr>pixdy1 & pixr>pixdy2) |...
              (pixr>pixdx1 & pixr>pixdx2));
resul= zeros(size(pixr)); resul(maxima)= pixr(maxima);
%=====
resul(:,1:2)=zeros(nlig,2); resul(1:2,:)=zeros(2,ncol);
resul(:,ncol-2:ncol)=zeros(nlig,3);
resul(nlig-2:nlig,:)=zeros(3,ncol);
resul=resul/max(resul(:))*Lcmap; % Normalization
return
```

The following program displays the obtained result on a test image (Figure 6.22):

```
%===== TRTEYES1.M
load eyes3;
[nlig, ncol]= size(pixc); Mcmap=size(cmap,1)-1;
%===== Differentiation
hd=dergauss(1); pixr=(filter2(hd,pixc));
%===== Looking for the maxima
pixr(:,1:3)=zeros(nlig,3); pixr(1:3,:)=zeros(3,ncol);
pixr(:,ncol-2:ncol)=zeros(nlig,3);
pixr(nlig-2:nlig,:)=zeros(3,ncol);
tbmax = searchmax(pixr,cmap);
subplot(131); imagesc(pixc+1); colormap(cmap); axis('image')
subplot(132); imagesc(pixr); axis('image')
subplot(133); imagesc(tbmax); axis('image')
save eyetst pixr tbmax cmap
```

The results are saved (`save` command) to the file `eyes3` for further processing.

Combining a Gaussian low-pass filter with first order horizontal and vertical derivatives, such as in the previous example, is a very common method for contour detection. J. Canny [18] showed that this method is very similar to applying a filter that optimizes a criterion related to precision and stability.

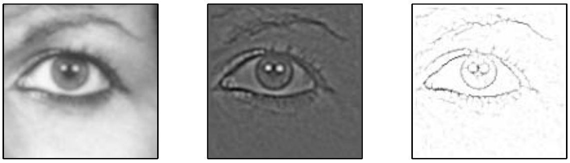


Figure 6.22 – *Results of thresholding and of the search for local maxima*

Exercise 6.9 (Contours using Sobel filtering)

1. Apply the Sobel filters 6.20 and 6.21 to the test image.

2. Using the resulting pixels $p_v(k,l)$ and $p_h(k,l)$, construct the image of the pixels $\sqrt{p_v^2(k,l)+p_h^2(k,l)}$. By defining an appropriate threshold value, extract the contours of the image.

The function `fminsearch` implementing the Nelder-Mead algorithm can be used for the seaching of extrema.

Detection of a given shape

Extracting contours is usually required to extract a given shape in an image. In Figure 6.22, it may be useful to properly detect an iris, for identification purposes for example. In order to achieve this detection, we are going to perform a filtering with a circular PSF, and we will work with the maxima of the results.

Exercise 6.10 (Iris search)
Using the array `tbmax` obtained in the previous program, imagine a way to identify where the iris is located.

Hough method

For shape recognition, it may be useful to detect the presence of basic shapes, such as circles, ellipses, straight lines, etc. The Hough method is one of the most common.

Consider for example the case of line detection in an image previously processed so as to outline the contours. For straight contour detection, we then use sets of lines where each straight line is defined by the pair of paramaters $(\rho,\ \theta)$ (Figure 6.23). ρ refers to the distance to the origin and θ the angle to the direction perpendicular to the line.

In each point of the contour, a set of concurrent lines is built, with the parameters ρ and θ (see Figure 6.23). Figure 6.24 shows that two sets of lines

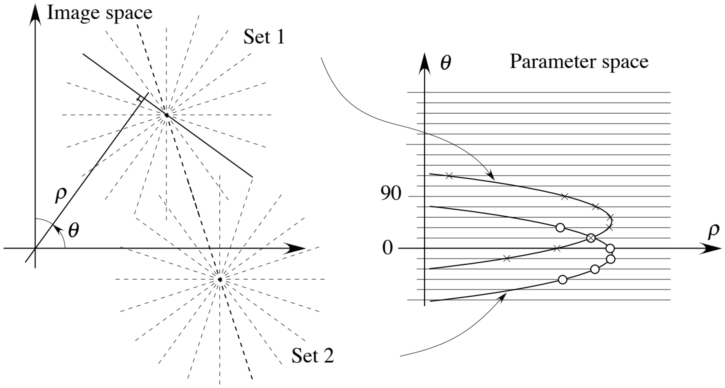


Figure 6.23 – *Setting the parameters for the set of lines*

on a portion of a line share a common point, or to be less specific, in the same neighborhood, on the parameters.

The accumulation, resulting from all the filters associated with this portion of a line, leads to a maximum in the neighborhood of this point. We then proceed to partitioning the parameter space, so as to obtain a quantization grid, then we count the points inside each box of the grid. The resulting values are then used for different kinds of processing.

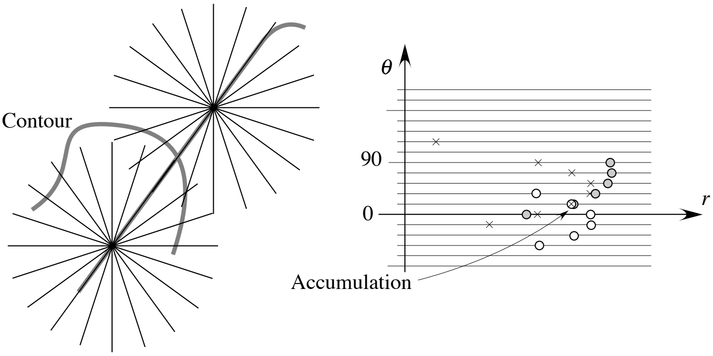


Figure 6.24 – *Using sets of lines*

Example 6.8 (Implementing the Hough method)
Consider an image (Figure 6.26, image on the left), assumed to have been obtained by contour extraction. The following program performs a search for straight lines

```
%===== HOUGH.M
load hough; [nlig,ncol]=size(pixc);
figure(1); subplot(121); colormap(cmap);
imagesc(pixc); axis('image')
%===== Contour extraction
indx=find(pixc==0); Nidx=length(indx);
%=====
Nt=60; thetad=[0:180/Nt:180]; theta=thetad*pi/180;
thet=thetad(1:Nt); tbl=zeros(Nt,Nt);
figure(2)
%===== For each point and each value
%      of theta, rho is computed
for k=1:Nidx
    nc=floor((indx(k)-1)/nlig)+1; nl=indx(k)-(nc-1)*nlig;
    for m=1:Nt, rho(m)=nc*cos(theta(m))+nl*sin(theta(m)); end
    tbl(:,k)=rho';
    plot(rho,thet); hold on
end
rhomax=sqrt(nlig*nlig+ncol*ncol);
set(gca,'Xlim',[0 rhomax]); grid; hold off
%===== Result (visual examination)
figure(1); subplot(122); imagesc(pixc); colormap(cmap);
axis('image'); hold on
plot([0 54.5*cos(70*pi/180)], [0 54.5*sin(70*pi/180)])
hold off
```

The observation of the resulting set (Figure 6.25) provides us with a direction.

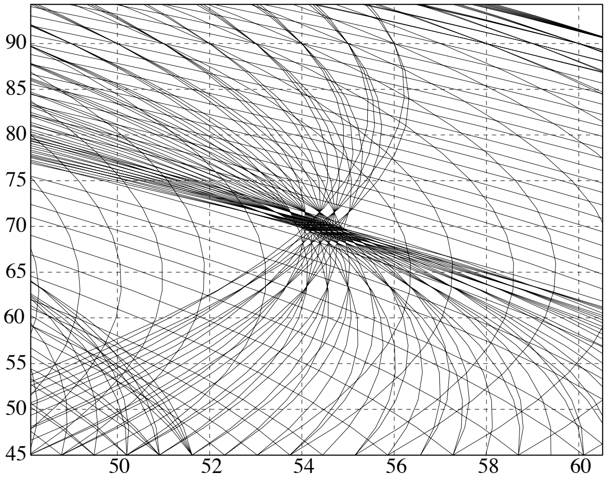


Figure 6.25 – *Zoom-in on the set of lines: we find the values* $\rho = 54.5$ *and* $\theta = 70^{\circ}$

The perpendicular direction is indicated in Figure 6.26 (the image on the right). An automatic search requires searching for zones with a high point density, hence the idea to use a 2D histogram to extract the $(\rho,\ \theta)$ positions of the maxima.

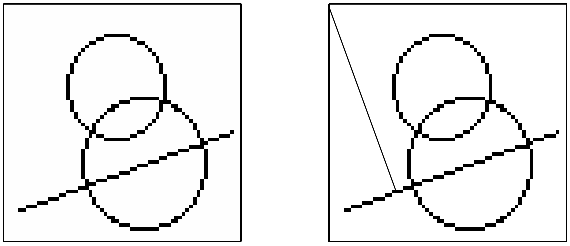


Figure 6.26 – *Search for a straight line*

One example of an application for this type of processing is the search for the writing line directions in a handwritten text. The method described here is known as the *Hough method* [46], or *Hough transform method*, and allows the extraction of directions, one of which still has to be chosen. The presence of a high point density along a line can help find them.

The Hough method can also be used to search for other shapes. The idea is the same. For example, for circular shapes, the parameter $\{r, \theta, R\}$ can be used, where the pair (r, θ) refers to the polar coordinates of the circle's center and R is its radius.

6.5.4 Median filtering

Compared to other non-linear filters, median filtering is both simple and efficient. Just like a linear low-pass filtering, it smooths the image and can therefore eliminate certain of the image's imperfections. However, unlike a linear low-pass filter, which inevitably adds a blur around the contours, it better preserves the sharp variations of the image.

Definition 6.10 (Median filter)
Let $\{a(k,\ell)\}$ *be an image. The median filter associates the mean value* $m(k,\ell)$ *with the point with coordinates* (k,ℓ)*, in the* $(M \times N)$ *rectangular window, centered on* (k,ℓ)*. If we assume* N *and* M *to be odd, and if* $u(n)$ *denotes the sorted sequence* $(u(n) \geq u(n-1))$ *obtained from the array* $[a(i,j)]$ $(M \times N)$ *where* $i \in \{(k-(M-1)/2, \ldots, k+(M-1)/2\}$ *and* $j \in \{(\ell-(N-1)/2, \ldots, \ell-(N+1)/2)\}$*, we have:*

$$m(k,\ell) = u((MN+1)/2)$$

Exercise 6.11 (Median filtering)
Apply this program to the test image:

```
%===== SNOWING.M
load lena
dims = size(pixc);
msnow = (randn(dims)>-2);
pixcmsnow = pixc .* msnow;
imagesc(pixcmsnow); axis('image')
```

This causes white points to randomly riddle the image, a bit like snow. Compare the effect of a Gaussian smoothing filter with the effect of a median filtering on the "snowy" image.

Exercise 6.12 (Processing the result of a rotation)
Use the saved image in example 6.2.

1. Perform a 3 × 3 median filtering on the resulting image. Try several rotation angles.
2. Perform a processing of the "missing" points by calculating a mean on the surrounding pixels.

There are many possible methods for processing an image after it has undergone geometric transformations: interpolations, morphological filtering (paragraph 6.5.7), median filterings...or other methods adapted to the case in question. There are no absolute rules in the field.

6.5.5 Maximum enhancement

The use of a unique threshold for the entire image can hide local maxima from view. Hence the idea to perform some transformations to enhance certain maxima before the thresholding. This non-linear processing significantly improves the detection of "peaks", in an image that has, more often than not, undergone a derivative and an extraction of local maxima.

An example of this type of processing is given in the following program which performs an affine transformation in order to bring the maxima to the same level:

```
function vecnorm=NormVec(vec,max0,seuil)
%%=============================================%
%% SYNOPSIS: vecnorm=NORMVEC(vec,max0,seuil)   %
%%    vec     = vector to be normalized        %
%%    max0    = global maximum                 %
%%    seuil   = threshold for the local maxima %
%%    vecnorm = global vector                  %
%%=============================================%
ip=find(vec<0); vec(ip)=zeros(1,length(ip));
```

```
Lvec=length(vec); tbvec=zeros(1,Lvec);
ip=find(vec>seuil); tbvec(ip)=ones(1,length(ip));
%=====
indbool0=1; indm=[];vm=[];
%===== Indices and values of the local maxima
for k=1:Lvec
    if (indbool0 & tbvec(k)),
        indbool0=0; k1=k;
    elseif(~indbool0 & ~tbvec(k)),
        [vmx,ivmx]=max(vec(k1:k-1));
        indbool0=1; indm=[indm ivmx+k1-1];
vm=[vm vmx];
    end
end
%=====
Lim=length(indm);
if Lim==0, vecnorm=vec; return; end
k1=indm(1); rapd=ones(1,k1)*max0/vm(1); rap=rapd;
k2=indm(Lim); rapf=ones(1,length(vec)-k2)*max0/vm(Lim);
rap=zeros(1,Lvec);
%=====
if Lim==1,
    rap=[rapd rapf];
else
    rap=rapd;
    for k=2:length(indm),
        i1=indm(k-1); i2=indm(k);
        y1=max0/vm(k-1); y2=max0/vm(k);
        for m=i1+1:i2,
            rap(m)=y1+(m-i1)*(y2-y1)/(i2-i1);
        end
    end
    rap=[rap rapf];
end
vecnorm=vec .* rap;
return
```

The following program uses the data taken from the search of maxima in an image (Figure 6.27):

```
%===== ENHANCEYE.M
load eyetst
subplot(221); imagesc(pixr); axis('image');
subplot(222); imagesc(tbmax); axis('image');
colormap(cmap)
[nlig, ncol]= size(pixr); Mcmap=size(cmap,1)-1;
%===== Improving the peaks with the derivative
max0=max(max(pixr)); [nlig, ncol]=size(pixr);
seuil=6; resul=zeros([nlig, ncol]);
for k=1:nlig;
```

```
    vecnorm=NormVec(pixr(k,:),max0,seuil);
    resul(k,:)=vecnorm;
end
subplot(223); imagesc(resul); axis('image');
%===== Improving the peaks
max0=max(max(tbmax)); [nlig, ncol]=size(tbmax);
seuil=45; resul2=zeros([nlig, ncol]);
for k=1:nlig;
    vecnorm=NormVec(tbmax(k,:),max0,seuil);
    resul2(k,:)=vecnorm;
end
subplot(224); imagesc(resul2); axis('image');
```

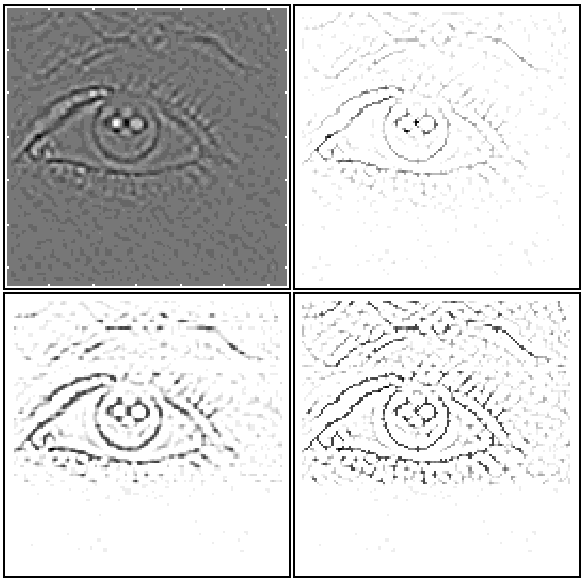


Figure 6.27 – *Processing by contour extraction*

6.5.6 Image binarization

Image binarization consists of intensifying the contrast until complete saturation is reached. Black and white are the only two levels kept after this operation. It is used in particular for Optical Character Recognition, or OCR. The technique described below is based on a method suggested by N. Otsu in 1979 [70]. It requires the calculation of an histogram first, followed by a separation in two categories, $\underline{\mathcal{C}}$ and $\overline{\mathcal{C}}$, associated with the two colors. It is quite simple to adapt this method to a greater number of categories:

1. The histogram calculation consists of initializing with zeros an array $\mathbf{H} = [h(k)]$ with $P = 256$ entries. These entries correspond to P levels of gray. The entire image is covered, and for each pixel (n, m) with a level of gray k, the entry for $h(k)$ in the array $\mathbf{H}$ is incremented. The histogram is normalized by dividing $\mathbf{H}$ by the number N of pixels in the image. The $h(k)$ can then be interpreted as estimated values for the probabilities of finding the 256 levels of gray in the image.

2. Separating the image pixels in two categories can be done by directly comparing levels of gray with a threshold value defined by observing the previous histogram.

This very simple method can give disappointing results. There are two main drawbacks. First, isolated pixels can belong to an area and not be part of that area's category. In particular, this can lead to highly contrasted textures. The second drawback concerns images showing the shadow of certain objects. It is not always a good thing to have them belong to the same category as the object they came from, whatever the lighting may be.

Consider for example the original image in Figure 6.29. The following program first draws the histogram for the 256 levels (Figure 6.28), then uses it to calculate threshold values. Based on these values, the program displays two binarization examples. The results, for two threshold values, are shown in Figure 6.29:

```
%===== BINAR1.M
load elido72
[nlig ncol]=size(pixc); nbpix=prod(size(pixc));
%===== Global histogram
histog=zeros(1,256); pixc3=zeros(nlig*ncol,1); pixc3(:)=pixc;
histog=hist(pixc3,256)/nlig/ncol;
figure(1); plot([0:255],histog); grid
%===== Thresholds based on a visual examination
%      of the histogram
figure(2); subplot(131);
imagesc(pixc); axis('image'); colormap(cmap);
%===== Threshold 1
pixc2=zeros(nlig,ncol);
seuil=152; idxy=find(pixc>seuil);
pixc2(idxy)=255*ones(size(idxy)); subplot(132); imagesc(pixc2);
axis('image'); colormap(cmap)
%===== Threshold 2
pixc2=zeros(nlig,ncol);
seuil=90; idxy=find(pixc>seuil);
pixc2(idxy)=255*ones(size(idxy)); subplot(133); imagesc(pixc2);
axis('image'); colormap(cmap)
save histog histog
```

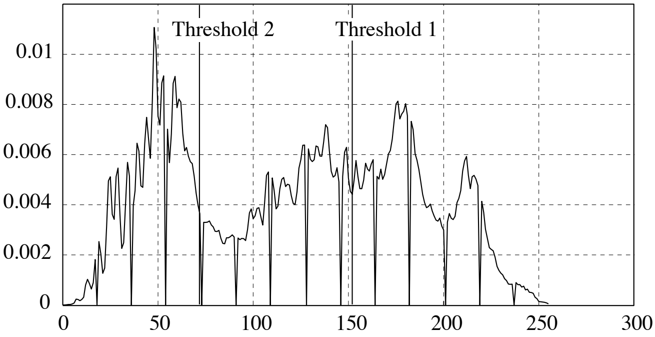


Figure 6.28 – *Histogram and global thresholds*

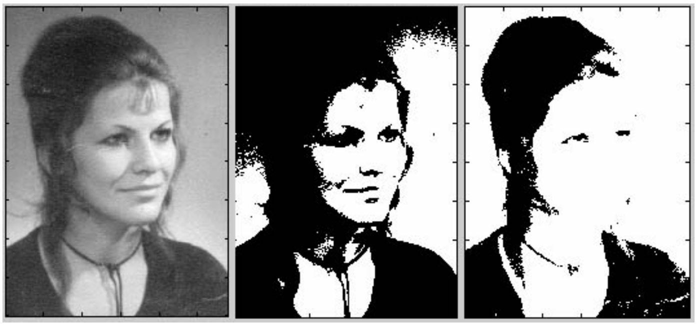


Figure 6.29 – *Binarization of the image above for two threshold values*

Automatic threshold calulation: the Otsu method

We will now see how to make the choice of the threshold automatic. In order to do this, we will write $h(k)$ to refer to the percentage of values from the image that are equal to k, where $k \in \{0, \ldots, 255\}$, as it was calculated in the previous histogram. $h(k)$ provides an *estimation for the probability* of level k.

Let s be the threshold. s defines two categories of values: category $\mathcal{C}_I$ for values below s, and category $\mathcal{C}_S$ for values above s. The method suggested by Otsu [70] simply consists of choosing, as the threshold value s, the integer for which the quadratic error is minimal between the observed random value k and its corresponding value $\mu(K)$ in one of the two categories.

This method is merely the particular case for 1 bit of the N scalar quantification problem, the solution of which is known as the Lloyd-Max solution.

Let μ_I and μ_S be the number of pixels in the categories $\mathcal{C}_I$ and $\mathcal{C}_S$ respectively. The expression of the criterion we wish to minimize, with respect to s,

μ_I and μ_S is:

$$L(s,\mu_I,\mu_S)=\sum_{k=0}^{s-1}(k-\mu_I)^2h(k)+\sum_{k=s}^{P-1}(k-\mu_S)^2h(k) \tag{6.26}$$

Minimizing this ratio as a function of μ_I and μ_S can be achieved by zeroing the partial derivatives of $L(s,\mu_I,\mu_S)$ with respect to μ_I and μ_S.

In the case of μ_I for example, this leads to:

$$\frac{\partial L(s,\mu_I,\mu_S)}{\partial \mu_I}=-2\sum_{k=0}^{s-1}(k-\mu_I)h(k)=0$$

the solution of which is:

$$\mu_I(s)=\frac{\sum_{k=0}^{s-1}kh(k)}{\sum_{k=0}^{s-1}h(k)} \tag{6.27}$$

Likewise, we have:

$$\mu_S(s)=\frac{\sum_{k=s}^{P-1}kh(k)}{\sum_{k=s}^{P-1}h(k)} \tag{6.28}$$

Notice that there is an obvious interpretation for μ_I and μ_S: they are the respective means of each category. By replacing these two expressions of $L(s,\mu_I,\mu_S)$ in 6.26, we get an expression $J(s)$, dependent only on s, which needs to be minimal. The solution cannot be obtained analytically, but the numerical solution can be found by calculating $J(s)$ for the 256 possible values of s.

There are two equivalent expressions of $J(s)$ that are best adapted for the numerical calculation. Let:

$$P_I(s)=\sum_{k=0}^{s-1}h(k) \text{ and } P_S(s)=\sum_{k=s}^{P-1}h(k) \tag{6.29}$$

such that $P_I(s)+P_S(s)=1$. The minimizing of $J(s)$ with respect to s is equivalent to *maximizing*:

$$G(s)=P_I(s)\mu_I^2(s)+P_S(s)\mu_S^2(s) \tag{6.30}$$

with respect to s. This is because:

$$\begin{aligned}J(s) &= \textstyle\sum_{k=0}^{s-1}k^2h(k)-2\sum_{k=0}^{s-1}k\mu_I(s)h(k)+\sum_{k=0}^{s-1}\mu_I^2(s)h(k)+\\ &\quad \textstyle\sum_{k=s}^{P-1}k^2h(k)-2\sum_{k=s}^{P-1}k\mu_S(s)h(k)+\sum_{k=s}^{P-1}\mu_S^2(s)h(k)\\ &= \textstyle\sum_{k=0}^{P-1}k^2h(k)-(\mu_I^2(s)P_I(s)+\mu_S^2(s)P_S(s))\end{aligned}$$

Hence, because the first term is independent of s, minimizing $J(s)$ is equivalent to maximizing $G(s) = \mu_I^2(s)P_I(s) + \mu_S^2(s)P_S(s)$.

The maximizing of $G(s)$ found in 6.30 is equivalent to the maximizing of:

$$H(s) = P_I(s)P_S(s)(\mu_I(s) - \mu_S(s))^2 \tag{6.31}$$

Notice that the quantity:

$$P_I(s)\mu_I(s) + P_S(s)\mu_S(s) = \sum_{k=0}^{P-1} k^2 h(k)$$

(see expressions 6.27, 6.28 and 6.29) is independent of s, we can calculate its square and substract this square value from $G(s)$ without changing the maximizing with respect to s. We get:

$$\begin{aligned} H(s) &= G(s) - (P_I(s)\mu_I(s) + P_S(s)\mu_S(s))^2 \\ &= \mu_I^2(s)P_I(s) + \mu_S^2(s)P_S(s) - \mu_I^2(s)P_I^2(s) - \mu_S^2(s)P_S^2(s) \\ &\qquad -2P_I(s)P_S(s)\mu_I(s)\mu_S(s) \\ &= \mu_I^2(s)P_I(s)(1 - P_I(s)) + \mu_S^2(s)P_S(s)(1 - P_S(s)) \\ &\qquad -2P_I(s)P_S(s)\mu_I(s)\mu_S(s) \end{aligned}$$

If we use $P_I(s) + P_S(s) = 1$, we get the expected result 6.31.

Exercise 6.13 (Application of the Otsu method)

1. Using MATLAB®, write the function that calculates the threshold obtained by maximizing expression 6.31.

2. Apply this function to a test image.

Figure 6.30 gives the result obtained by calculating the threshold with the Otsu method.

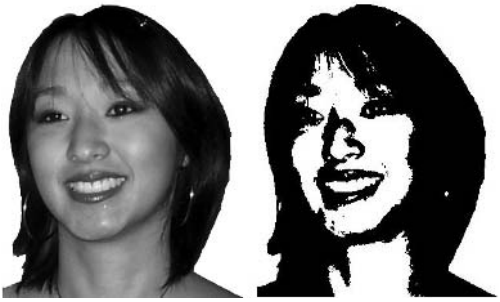


Figure 6.30 – *Binarization for the threshold calculated with the Otsu method*

6.5.7 Morphological filtering of binary images

A morphological filtering is a filtering that uses `min` and `max` operations. This can be symbolized as follows:

$$p_{k,l} = \mathcal{F}\left(\mathcal{B}(P)\right) \tag{6.32}$$

where P is an image, $\mathcal{B}(P)$ a portion of the image extracted using a window $\mathcal{B}$, and $\mathcal{F}$ is a logical operation applied to pixels isolated by the window $\mathcal{B}$. The following function, called *erosion*, illustrates the process 6.32 applied to a binary image when the `min` operation amounts to a logical AND:

```
function ppx=erosion(block,mtool)
%%===================================================%
%% SYNOPSIS: ppx=EROSION(block,mtool)                %
%%   block = Data block of the same size as the tool %
%%   mtool = Matrix defining the tool shape          %
%%   Example: [0 1 0;1 1 1;0 1 0] defines a cross.   %
%%   ppx   = Resulting pixel value                   %
%%===================================================%
L=round(log(size(colormap,1))/log(2));
ido=find(mtool==1); Lido=length(ido);
ppx=block(ido(1));
for m=2:Lido,
    pps=block(ido(m)); ppt=ppx; st=0;
    for k=1:L
        st=st + (rem(ppt,2) & rem(pps,2)) * 2^(k-1);
        ppt=fix(ppt/2); pps=fix(pps/2);
    end
    ppx=st;
end
return
```

In this function, the windowing matrix associated with the operator $\mathcal{B}$ is referred to as the *structuring element*. It consists of a boolean matrix. A “1” indicates a pixel that needs to be taken into account by the logical function processing. Hence the processing can be symbolized by:

$$p_{k,l} = \bigcap_{\{n,m:b_{n,m}=1\}} p_{n,m}$$

The program `exerosion.m` illustrates the `erosion` function call:

```
%===== EXEROSION.M
clear
pixres=raw2matf('exerode.raw',128,128,'T');
cmap=[1:-1/255:0]'*[1 1 1]; [nl0,nc0]=size(pixres);
subplot(121); colormap(cmap); NbLevel=size(cmap,1);
imagesc(pixres); axis('image')
```

```
%===== Defining the window
Nlig=1; Ncol=1; mtool=ones((2*Nlig+1),(2*Ncol+1));
%===== The image must be coded between 0 and NbLevel-1
pixc=[ones(nl0,Ncol) (pixres-1) ones(nl0,Ncol)];
pixc=[ones(Nlig,nc0+2*Ncol);pixc;ones(Nlig,nc0+2*Ncol)];
subplot(122); imagesc(pixc); axis('image')
pixr=zeros(nl0,nc0);
%=====
for nl=Nlig+1:Nlig+nl0
    for nc=Ncol+1:Ncol+nc0
        blk=pixc(nl-Nlig:nl+Nlig,nc-Ncol:nc+Ncol);
        ppx(nl-Nlig,nc-Ncol)=erosion(blk,mtool);
    end
end
subplot(122); imagesc(ppx); axis('image')
```

In this program, the structuring element is a (3×3) square. Its execution is particularly slow. MATLAB® is not well suited for this type of processing comprising many loops. The best method would once again be to write a dedicated "`.mex`" function. The image *toolbox*, of course, provides such functions.

If, in the `erosion.m` function, the AND ($\cap$) function is replaced with an OR ($\cup$) function, the result is a *dilation* function. This means we are dealing with the implementation of the `max` function for binary images.

Figure 6.31 illustrates the respective effects of erosion and dilation. In the case of erosion, any pattern not covered by the window disappears. The contours of the objects in the foreground are "eroded". Dilation, on the contrary, emphasizes the image's details by "increasing" their size.

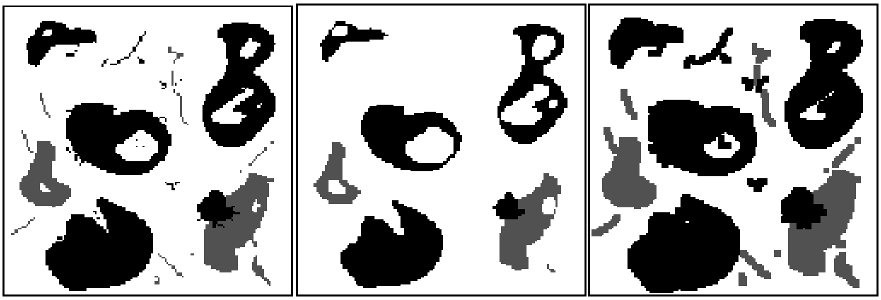


Figure 6.31 – *Effects of erosion and dilation: original, eroded and dilated images from left to right respectively*

6.6 JPEG lossy compression

The JPEG format (*Joint Photographic Experts Group*) for coding image files is widely used because of the compression rates it can achieve without significant quality loss. We are going to construct the functions of this coding, without trying, however, to construct the final binary flux.

The idea behind this coding has to do with the use of the discrete cosine transform, or DCT.

6.6.1 Basic algorithm

The JPEG compression (lossy compression) algorithm can be very briefly summed up as follows [42]:

- the image is divided in blocks of 8 by 8 pixels, to which a DCT is applied (the blocks are read line by line, from top to bottom and from left to right) The basic process implies that the levels associated with each pixel are 8 bit coded. To make things simpler, we will assume that the images we are going to process are given in "levels of gray";
- the 64 coefficients of the DCT are quantified (rounded);
- the "mean value" (DCT value at the frequency 0) is subtracted from the same term of the next block;
- the 63 other terms are read in "zigzags" (Figure 6.32);

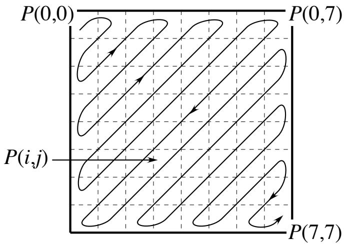


Figure 6.32 – *Reading of the DCT coefficients*

- the sequence of the obtained values is coded (Huffman entropic coding);
- each non-zero coefficient is coded by the number of zeros preceding it, the number of bits needed for its coding, and its value. The coding rules are imposed by the [93] standard.

We assume that we will keep the floating-point representation coding. We will not try to optimize the size of the coded DCTs.

6.6.2 Writing the compression function

Writing the DCT calculation and quantification functions

Consider an (8×8) array of pixels $p(x, y)$ $(x, y \in \{0, \ldots, 7\})$, the DCT's expression for $u, v \in \{0, \ldots, 7\}$:

$$P(u, v) = \frac{1}{4} C(u) C(v) \sum_{x=0}^{7} \sum_{y=0}^{7} p(x, y) \cos \frac{(2x+1)\pi u}{16} \cos \frac{(2y+1)\pi v}{16} \quad (6.33)$$

with $C(0) = \frac{1}{\sqrt{2}}$ and $C(k) = 1$ for $k = 1 \ldots 7$. Once the coefficients are obtained, the array is weighted and quantified:

$$P_q(u, v) = \texttt{round} \frac{P(u, v)}{Qtab(u, v)}$$

where *Qtab* is a quantification table for chrominance included with the standard as an example. It is supposed [99] to provide good results for the type of coding performed here and for most images commonly dealt with.

The following initialization function returns the *Qtab* table as well as the indices used for the "zigzag" reading of the DCT array:

```
function [Qtab,zig,zag]=initctes
%%===========================================%
%% Init. of the constants for the JPEG algorithm %
%%===========================================%
global mNORM mYV mUX
mUX = cos([0:7]'*(2*[0:7]+1)*pi/16);
mYV = cos((2*[0:7]'+1)*[0:7]*pi/16);
mNORM = [1/2 ones(1,7)/sqrt(2); ones(7,1)/sqrt(2) ones(7,7)]/4;
Qtab=[16 11 10 16 24 40 51 61;
      12 12 14 19 26 58 60 55;
      14 13 16 24 40 57 69 56;
      14 17 22 29 51 87 80 62;
      18 22 37 56 68 109 103 77;
      24 35 55 64 81 104 113 92;
      49 64 78 87 103 121 120 101;
      72 92 95 98 112 100 103 99];
zig=[1 9 2 3 10 17 25 18 ...
     11 4 5 12 19 26 33 41 ...
     34 27 20 13 6 7 14 21 ...
     28 35 42 49 57 50 43 36 ...
     29 22 15 8 16 23 30 37 ...
     44 51 58 59 52 45 38 31 ...
     24 32 39 46 53 60 61 54 ...
     47 40 48 55 62 63 56 64];
zag=zig(64:-1:1);
return
```

Exercise 6.14 (Writing basic functions)

1. Write the calculation function of the DCT using the vectors `mNORM`, `mYV` and `mUX`, which will be declared `global mNORM mYV mUX` and initialized with the `initctes.m` function.

2. Write the quantification function.

3. Check the processing using the following data [2].

The data we are working with are coded on one byte, a value between 0 and 255, that you need to bring back between -128 and 127:

```
%===== DATAEX.M
pix=[139 144 149 153 155 155 155 155;
     144 151 153 156 159 156 156 156;
     150 155 160 163 158 156 156 156;
     159 161 162 160 160 159 159 159;
     159 160 161 162 162 155 155 155;
     161 161 161 161 160 157 157 157;
     162 162 161 163 162 157 157 157;
     162 162 161 161 163 158 158 158];
```

The result has to be:

```
    15     0    -1     0     0     0     0     0
    -2    -1     0     0     0     0     0     0
    -1    -1     0     0     0     0     0     0
    -1     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

COMMENT: the quantification table is usually associated with a *quality factor* F_q. The previous table corresponds to $F_q = 50$ %. The following function can be used to generate tables for other values of F_q:

```
function [Qtab]=TabQuantif(Fq)
%%======================================%
%% SYNOPSIS: [Qtab]=TABQUANTIF(Fq)      %
%%     Fq   = Quality factor (0 to 100) %
%%     Qtab = Weighting table           %
%%======================================%
if nargin<1, Fq=50; end
%===== Table for a quality factor = 50
Qtab=[16 11 10 16 24 40 51 61 ;
      12 12 14 19 26 58 60 55 ;
      14 13 16 24 40 57 69 56 ;
      14 17 22 29 51 87 80 62 ;
      18 22 37 56 68 109 103 77 ;
      24 35 55 64 81 104 113 92 ;
      49 64 78 87 103 121 120 101 ;
      72 92 95 98 112 100 103 99];
%=====
if (Fq<50)
   scal = 5000/Fq; else scal = 200 - Fq*2;
end
Qtabnew=floor(((Qtab.*scal)+50)./100);
idz=find(Qtabnew<=0); Qtabnew(idz)=ones(size(idz));
idz=find(Qtabnew>255); Qtabnew(idz)=255*ones(size(idz));
Qtab=Qtabnew;
return
```

Exercise 6.15 (Writing the compressed frame)

1. Using the functions written in exercise 6.14, write the function that creates the compressed frame for *one block*. The starting mean value is assumed to be zero. For the previous exercise, the result, with some comments, should be:

```
%==== Test block
5         number of bits used to code the difference
15        difference with the previous block's mean (0 in this case)
1,2,-2    1 zero before the -2, which is 2 bit coded
0,1,-1    No zero before the -1 which is 1 bit coded
0,1,-1    Idem
0,1,-1    Idem
2,1,-1    2 zeros before the -1 which is 1 bit coded
0,1,-1    No zero before the -1 which is 1 bit coded
0,0       There is nothing left but zeros
```

   Do not calculate, for now, the number of bits needed for coding each of the DCT's coefficients. We will assume it is the same for all of them, and that its value is 17.

   Save the compressed data to the file `unbloccode.dat`, but after having added at the beginning of the file the number of line blocks and of column blocks as follows:

```
fid=fopen('unbloccode.dat','w');
fwrite(fid,nby,'integer*1');
fwrite(fid,nbx,'integer*1');
% Writing the compressed data
for k=1:...
    fwrite(fid,...,'integer*1');
end
fclose(fid);
```

2. Apply the obtained program to the test image. Save the compressed data to the file `imgtstcode.dat`.

6.6.3 Writing the decompression function

Inverse DCT

The inverse DCT, referred to as the ICDT, is given by 6.34:

$$p(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} P(u,v)C(u)C(v)\cos\frac{(2x+1)\pi u}{16}\cos\frac{(2y+1)\pi v}{16} \tag{6.34}$$

Exercise 6.16 (Decompression)

1. Using the *Qtab* table given on page 237, write the "dequantization function" of the DCT coefficients.

2. Write the inverse DCT function.

3. Test the "decompression" operation by applying it to the previously used test block which is coded as follows:

```
A=[1,1,17,15,1,17,-2,0,17,-1,0,17,-1,0, ...
   17,-1,2,17,-1,0,17,-1,0,0]
```

   The first two terms indicate that there is only one block.

4. Apply the program to the file `imgtstcode.dat` obtained in exercise 6.15 (the result obtained with the test image that was chosen is shown in Figure 6.33).

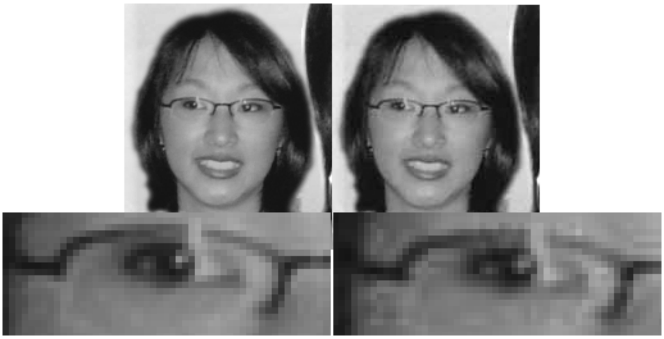


Figure 6.33 – *Comparing original images with images obtained by coding and decoding for a quality factor of* $\approx 30\%$

6.7 Watermarking

Having access to digitalized information such as images, sound or video raises important issues regarding intellectual property and copyright. Those who edit these contents have a strong demand for systems that can protect or at least identify the documents that can be easily downloaded off of a network. Watermarking is a collection of methods used for leaving a mark on these documents. We will restrict ourselves to still images, and only give a few ideas on the techniques that are used.

The difficulty is due to the fact that the watermarking must satisfy several contradictory constraints: the mark left on the document must be both not too visible (to a certain extent) and easy to reconstruct. This reconstruction must however be easy only for the one who left the mark. Furthermore, the mark must not be destroyed by the manipulations an image can undergo when it travels (coding and decoding), when it is stored (compression), when it sustains an ill-intentioned process or several successive markings.

Watermarks can be fragile or robust:

1. A *fragile watermark* implies that the watermark reconstruction will be perfect. It makes it possible to know whether or not the image was altered. This method is needed by users who wish to be certain that the documents they receive are originals.

2. A *robust watermark* must make it possible to know if there has been a watermark despite any attempt to destroy it, and must preserve the copyright information attached to the document.

The following pages give a few ideas for watermarking methods. You can now find articles (see [103] for example) that sum up all of the methods that are used.

6.7.1 Spatial image watermarking

Spatial watermarking consists of adding a mark to the image, visible or invisible, made up of another image, such as a logo, or a secret key that only the image's creator has.

Example 6.9 (Use of the least significant bits (LSB))
We are going to start by performing a highly fragile deterministic marking. The mark is the image of a stamp added to the least significant bits for levels of gray. The following program adds the image of a stamp to the original "lena" image. We voluntarily chose to alter four bits in order to see the stamp in the final image (Figure 6.34). It is impossible, of course, to perfectly reconstruct the original in this case. In practice, because the stamp is coded on two levels, it would have been enough to alter only one bit, and the transformation would be invisible to the naked eye:

```
%===== WM01.M
load lena; subplot(221)
imagesc(pixc); axis('image'); colormap(cmap)
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
%===== NbBits trailing bits set to zero
NbBits=4; nbniv=2^NbBits-1;
mmask=(255-nbniv) * ones(size(pixc));
pixcm=FoncLog(pixc, '&', mmask);
subplot(222); imagesc(pixcm); axis('image'); colormap(cmap)
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
%===== Mark
load tampon; subplot(223);
imagesc(pixct); axis('image'); colormap(cmap)
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
mmark=round((pixct*nbniv)/255);    % Level adjustment
pixcmm=FoncLog(pixcm, '|', mmark); %
subplot(224); imagesc(pixcmm); axis('image'); colormap(cmap)
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
```

Notice that the levels of gray of the mark are changed to adequate values, so as to allow them to be included in the least significant bits of the image (level adjustment).

Easy to implement, this method leads, however, to a watermark that is not in the least robust in regard to compression and noising operations. Also, it is possible for any user to reconstruct the mark.

Example 6.10 (The IBM method)
This method [65] was implemented at the Vatican's request in order to certify the origin of the works made available to everybody on the Internet. It consists of inserting in the image a watermark that is both visible and reversible. The color alteration algorithm is known only to IBM, and it is the only tool that can be used to "wash away" the watermark image.

Example 6.11 (The patchwork method)
This method consists of choosing pairs of pixels based on a random selection. One of the two is made brighter, and the other darker. This defines a transition used to code a watermark.

Exercise 6.17 (Yeung and Wong method)
This method makes it more difficult to reconstruct the mark than it was in example 6.9. First, we generate a key, allowing us to define a function that associates a 0 or 1 value with each level of gray:

$$\{0, 1, \ldots, 255\} \xrightarrow{f} \{0, 1\}$$

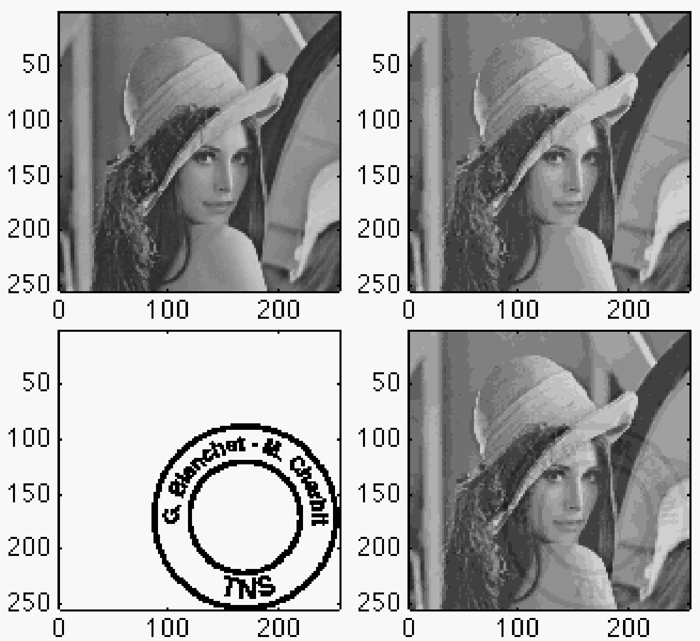


Figure 6.34 – *Watermark using the least significant bits of the image. The top-left image is the original. The top right image is the one with its four least significant bits set to zero. The bottom-right image is the one with the "stamp"*

Consider a mark $\mathcal{M}$ made up of $(N \times M)$ pixels coded on two levels $m_{k,l}$. An $(N \times M)$ portion of the original image $\mathcal{I}$. Let $i_{k,l}$ be the level of gray value of a pixel from $i_{k,l}$, and let $i'_{k,l}$ be its modified value. Necessarily, $f(i'_{k,l}) = m_{k,l}$.

Knowing the function f, or in other words knowing the key, allows you to reconstruct the original image. The point of the following exercise is to implement this method.

1. Define a binary mark (two levels only).

2. Using the `rand` function, generate the function f and save it. Write the image modification program.

3. Write the program designed to reconstruct the mark based on the modified image.

Other methods consist of adding a secret key to a group of pixels chosen in the image. Each pixel contains only one bit of the key (the *Walton* method).

It is also possible to generate a random bipolar sequence (comprised of 1 and -1) which is added to each of the lines of the image, with a translation that changes from one line to the next (the *van Schyndel* method). Correlation methods are used to search for the presence or the absence of the mark.

6.7.2 Spectral image watermarking

Watermarks based on frequency considerations can be justified by the following comments:

- Using a transform (Fourier, Gabor, DCT, wavelets) to generate the mark allows it to be stretched, in the transformed area, over the entire image. This makes it harder to locate it.
- It is easier to include perceptual considerations in the method that is chosen.
- When using the DCT, the result shows good robustness when it undergoes compressions of the JPEG type.

Exercise 6.18 (DCT modulation)

The method implemented in this exercise makes it possible to confirm an image's authenticity, without visibly altering it.

1. Write the direct and inverse DCT calculation functions for any image size using the expressions found for the DCT in the 8×8 case.
2. Write a program that performs the following operation:
   (a) calculates an image's DCT. Sorts the coefficients by modulus value. Keeps only N of these coefficients, N depending on the percentage P of the power contained in this set of coefficients. Let c_k, $k = 1 \ldots N$ be these coefficients;
   (b) generates a binary bipolar (-1 and $+1$) sequence w_k of length $N-1$;
   (c) adds the c_k, $k = 2 \ldots N$, and the w_k, $k = 1 \ldots N-1$.
3. Write a watermark identification program, knowing that you have at your disposal the original image and the watermark.

Part II

Random Signals

This page intentionally left blank

Chapter 7

Random Variables

7.1 Random phenomena in signal processing

In many practical circumstances, the phenomena observed show important variations, when in fact the relevant information itself has not changed. Thus, if you record the signals obtained when pronouncing several times the sound "A", a simple observation will tell you that all the recordings are different even though they all sound basically the same. It is neither easy nor relevant to try to find a deterministic equation to describe the evolution in time of such a phenomenon. The model used to describe this variability is based on the concept of *random variables* (r.v.), defined by the *Probability Theory.* A common misuse of language consists of saying that this is a *random* phenomenon.

Another example of a random phenomenon is the background noise heard with radio reception. It seems difficult to describe this noise without using statistical characteristics. In the complete chain of communications, an example we will come back to later, every device, as well as the transmission medium, causes background noise. But with such a system, this is not the only "source of randomness". From the receiver's point of view, the message itself must also be considered random.

In fact, any device or physical phenomenon has a random part to it. Deciding how important that random part is to the system is the only factor in determining what type of model is used to describe the phenomena. If the amount of information that we cannot have knowledge of is negligible, then we will choose a deterministic approach. The evolutionary models this leads us to are comprised of differential or recursive equations, analytical expressions, etc. In the opposite case, we need to use probabilistic models to try to represent the variability of the observed signal with a time-indexed sequence of random variables. Each of the random variable describes the uncertainties related to the phenomenon *at a given time.* A family of random variables is called a *random process.* They will be studied in Chapter 8, but we will first give a quick overview of the main properties of random variables.

7.2 Basic concepts of random variables

Without describing in detail the formalism used by the Probability Theory, we will simply remind the reader that a random variable is an application that associates a numerical value with each possible outcome of a trial.

A familiar image is the presence of values between 1 and 6 in a trial consisting of rolls of a dice. However, to be comfortable enough with probabilistic tools, we need to go beyond this simple definition. This is why we advise the reader to consult some of the many books with authority on the subject [28, 12].

From a practical point of view, it is often enough to distinguish two situations, that is whether the set of possible values of the random experiment is *discrete* or *continuous*. The number of people waiting in a line is an example of the discrete situation: the only possible values are zero or positive integers. Whereas taking down the speeds of vehicles on a road is an example of a continuous random variable: this time, the possible values are real numbers between 0 and 65 mph.

Definition 7.1 (Discrete random variable)
A random variable X is said to be discrete if the set of its possible values is, at the most, countable. *If $\{a_0, \ldots, a_n, \ldots\}$, where $n \in \mathbb{N}$, is the set of its values, the* probability distribution *(p.d.) of X is characterized by the sequence:*

$$p_X(n) = \Pr(X = a_n) \tag{7.1}$$

representing the probability that X is equal to the element a_n. These values are such that $0 \leq p_X(n) \leq 1$ and $\sum_{n\geq 0} p_X(n) = 1$.

This leads us to the probability for the random variable X to belong to the interval $]a, b]$. It is given by:

$$\Pr(X \in]a,b]) = \textstyle\sum_{n\geq 0} p_X(n)\mathbb{1}(a_n \in]a,b])$$

The function defined for $x \in \mathbb{R}$ by:

$$\begin{aligned} F_X(x) &= \Pr(X \leq x) = \textstyle\sum_{\{n: a_n \leq x\}} p_X(n) \\ &= \textstyle\sum_{n\geq 0} p_X(n)\mathbb{1}(a_n \in]-\infty, x]) \end{aligned} \tag{7.2}$$

is called the *cumulative distribution function* (*cdf*) of the random variable X. It is a monotonic increasing function, and verifies $F_X(-\infty) = 0$ and $F_X(+\infty) = 1$. Its graph resembles that of a staircase function (see Figure 7.6), the jumps of which are located at x-coordinates a_n and have an amplitude of $p_X(n)$.

Definition 7.2 (Two discrete random variables)
Let X and Y be two discrete random variables, with possible values $\{a_0, \ldots, a_n, \ldots\}$ and $\{b_0, \ldots, b_k, \ldots\}$ respectively. The joint probability distribution is characterized by the sequence of positive values:

$$p_{XY}(n,k) = \Pr(X = a_n, Y = b_k) \tag{7.3}$$

with $0 \leq p_{XY}(n,k) \leq 1$ and $\sum_{n \geq 0} \sum_{k \geq 0} p_{XY}(n,k) = 1$.

$\Pr(X = a_n, Y = b_k)$ represents the probability to *simultaneously* have $X = a_n$ and $Y = b_k$. This definition can easily be extended to the case of a finite number of random variables.

Property 7.1 (Marginal probability distribution) *Let X and Y be two discrete random variables, with possible values $\{a_0 \ldots a_n \ldots\}$ and $\{b_0 \ldots b_k \ldots\}$ respectively, and with their joint probability distribution characterized by $p_{XY}(n,k)$. We have:*

$$\begin{aligned} p_X(n) = \Pr(X = a_n) &= \sum_{k=0}^{+\infty} p_{XY}(n,k) \\ p_Y(k) = \Pr(Y = b_k) &= \sum_{n=0}^{+\infty} p_{XY}(n,k) \end{aligned} \tag{7.4}$$

$p_X(n)$ and $p_Y(k)$ denote the marginal probability distribution of X and Y respectively.

Definition 7.3 (Continuous random variable)
A random variable is said to be continuous if its values belong to $\mathbb{R}$ and if, for any real numbers a and b, the probability that X belongs to the interval $]a, b]$ is:

$$\Pr(X \in]a,b]) = \int_a^b p_X(x)dx \tag{7.5}$$

where $p_X(x)$ is a function that must be positive or equal to zero (not necessarily less than 1) such that $\int_{-\infty}^{+\infty} p_X(x)dx = 1$. $p_X(x)$ is called the probability density function (pdf) of X.

The function defined for any $x \in \mathbb{R}$ by:

$$F_X(x) = \Pr(X \leq x) = \int_{-\infty}^{x} p_X(u)du \tag{7.6}$$

is called the *cumulative distribution function* (cdf) of the random variable X. It is a monotonic increasing function and it verifies $F_X(-\infty) = 0$ and $F_X(+\infty) = 1$. Notice that $p_X(x)$ also represents the derivative of $F_X(x)$ with respect to x.

Definition 7.4 (Two continuous random variables)
Let X and Y be two random variables with possible values in $\mathbb{R} \times \mathbb{R}$. They are said to be continuous if, for any domain Δ of $\mathbb{R}^2$, the probability that the pair (X, Y) belongs to Δ is given by:

$$\Pr((X,Y) \in \Delta) = \iint_{\Delta} p_{XY}(x,y)dxdy \tag{7.7}$$

where the function $p_{XY}(x,y) \geq 0$, and is such that:

$$\iint_{\mathbb{R}^2} p_{XY}(x,y)dxdy = 1$$

$p_{XY}(x,y)$ is called the joint probability density function of the pair (X,Y).

Property 7.2 (Marginal probability distributions) *Let X and Y be two continuous random variables with a joint probability distribution characterized by $p_{XY}(x,y)$. The probability distributions of X and Y have the following marginal probability density functions:*

$$\begin{aligned} p_X(x) &= \int_{-\infty}^{+\infty} p_{XY}(x,y)dy \qquad (7.8) \\ p_Y(y) &= \int_{-\infty}^{+\infty} p_{XY}(x,y)dx \end{aligned}$$

The marginal probability density functions of X and Y are referred to as $p_X(x)$ and $p_Y(y)$.

An example involving two real random variables (X, Y) is the case of a complex random variable $Z = X + jY$.

It is also possible to have a mixed situation, where one of the two variables is discrete and the other is continuous. This leads to the following:

Definition 7.5 (Mixed random variables)
Let X be a discrete random variable with possible values $\{a_0 \ldots a_n \ldots\}$ and Y a continuous random variable with possible values in $\mathbb{R}$. For any value a_n, and for any real numbers a and b, the probability:

$$\Pr(X = a_n, Y \in]a,b]) = \int_a^b p_{XY}(n,y)dy \tag{7.9}$$

where the function $p_{XY}(n,y)$, with $n \in \{0 \ldots k \ldots\}$ and $y \in \mathbb{R}$, is ≥ 0 and verifies $\sum_{n \geq 0} \int_{\mathbb{R}} p_{XY}(n,y)dy = 1$.

Definition 7.6 (Two independent random variables)
Two random variables X and Y are said to be independent if and only if their joint probability distribution is the product of the marginal probability distributions. This can be expressed (for the previous cases only):

- *For two discrete random variables:*

$$p_{XY}(n,k) = p_X(n)p_Y(k)$$

- *For two continuous random variables:*

$$p_{XY}(x,y) = p_X(x)p_Y(y)$$

- *For two mixed random variables:*

$$p_{XY}(n,y) = p_X(n)p_Y(y)$$

where the marginal probability distributions are obtained with formulae 7.4 and 7.8.

We wish to insist on the fact that, knowing $p_{XY}(x,y)$, we can tell whether or not X and Y are independent. To do this, we need to calculate the marginal probability distributions and to check that $p_{XY}(x,y) = p_X(x)p_Y(y)$. If that is the case, then X and Y are independent.

The following definition is more general.

Definition 7.7 (Independent random variables) *The random variables $(X_1,\ldots,X_n)$ are jointly independent if and only if their joint probability distribution is the product of their marginal probability distributions. This can be expressed:*

$$p_{X_1X_2\cdots X_n}(x_1,x_2,\ldots,x_n) = p_{X_1}(x_1)p_{X_2}(x_2)\ldots p_{X_n}(x_n) \tag{7.10}$$

where the marginal probability distributions are obtained as integrals with respect to $(n-1)$ variables, calculated from $p_{X_1X_2\ldots X_n}(x_1,x_2,\ldots,x_n)$.

For example, the marginal probability distribution of X_1 has the expression:

$$p_{X_1}(x_1) = \underbrace{\int\ldots\int}_{\mathbb{R}^{n-1}} p_{X_1X_2\ldots X_n}(x_1,x_2,\ldots,x_n)dx_2\ldots dx_n$$

In practice, the following result is a simple method for determining whether or not random variables are independent:

If $p_{X_1X_2\ldots X_n}(x_1,x_2,\ldots,x_n)$ is a product of n positive functions of the type $f_1(x_1)f_2(x_2)\ldots f_n(x_n)$, then the variables are independent.

It should be noted that if n random variables are independent of one another, it does not necessarily mean that they are jointly independent.

Definition 7.8 (Mathematical expectation)
Let X be a random variable and $f(x)$ a function. The mathematical expectation of $f(X)$ (respectively $f(X,Y)$) is the value, denoted by $\mathbb{E}\{f(X)\}$ (respectively $\mathbb{E}\{f(X,Y)\}$), defined:

- *For a discrete random variable, by:*

$$\mathbb{E}\{f(X)\} = \sum_{n\geq 0} f(a_n)p_X(n)$$

- *For a continuous random variable, by:*

$$\mathbb{E}\{f(X)\} = \int_{\mathbb{R}} f(x)p_X(x)dx$$

- *For two discrete random variables, by:*

$$\mathbb{E}\{f(X,Y)\} = \sum_{n\geq 0}\sum_{k\geq 0} f(a_n, b_k)p_{XY}(n,k)$$

- *For two continuous random variables, by:*

$$\mathbb{E}\{f(X,Y)\} = \int_{\mathbb{R}}\int_{\mathbb{R}} f(x,y)p_{XY}(x,y)dxdy$$

Property 7.3 *If $\{X_1,\ X_2,\ \ldots,\ X_n\}$ are jointly independent, then for any integrable functions $f_1,\ f_2,\ \ldots,\ f_n$:*

$$\mathbb{E}\left\{\prod_{k=1}^{n} f_k(X_k)\right\} = \prod_{k=1}^{n}\mathbb{E}\{f_k(X_k)\} \tag{7.11}$$

Definition 7.9 (Characteristic function)
The characteristic function of the probability distribution of the random variables $X_1,\ \ldots,\ X_n$ is the function of $(u_1,\ldots,u_n) \in \mathbb{R}^n$ defined by:

$$\phi_{X_1\cdots X_n}(u_1,\cdots,u_n) = \mathbb{E}\left\{e^{ju_1X_1+\cdots+ju_nX_n}\right\} = \mathbb{E}\left\{\prod_{k=1}^{n} e^{ju_kX_k}\right\} \tag{7.12}$$

Because $\left|e^{juX}\right| = 1$, the characteristic function exists and is continuous even if the moments do not exist. The Cauchy probability distribution, for example, the probability density function of which is $p_X(x) = 1/\pi(1+x^2)$, has no moment and has the characteristic function $e^{-|u|}$. Notice that $|\phi_{X_1\cdots X_n}(u_1, \cdots, u_n)| \leq \phi_X(0, \cdots, 0) = 1$.

Theorem 7.1 (Fundamental) *$(X_1, \cdots, X_n)$ are independent if and only if for any point $(u_1, u_2, \ldots, u_n)$ of $\mathbb{R}^n$:*

$$\phi_{X_1\cdots X_n}(u_1, \cdots, u_n) = \prod_{k=1}^{n} \phi_{X_k}(u_k)$$

Notice that the characteristic function $\phi_{X_k}(u_k)$ of the marginal probability distribution of X_k can be directly calculated using 7.12. We have $\phi_{X_k}(u_k) = \mathbb{E}\left\{e^{ju_kX_k}\right\} = \phi_{X_1\cdots X_n}(0, \cdots, 0, u_k, 0, \cdots, 0)$.

Example 7.1 (First calculations)
Let X be a random variable with possible values in $\{0, 1\}$ with $\Pr(X = 0) = p_0 \geq 0$, $\Pr(X = 1) = p_1 \geq 0$ and $p_0 + p_1 = 1$. Calculate $\mathbb{E}\{X\}$, $\mathbb{E}\{X^2\}$, $\mathbb{E}\{\cos(\pi X)\}$ and $\phi_X(u)$.

HINT: we get:

$$\mathbb{E}\{X\} = 0 \times p_0 + 1 \times p_1 = p_1 \text{ and } \mathbb{E}\left\{X^2\right\} = 0^2 \times p_0 + 1^2 \times p_1 = p_1$$

then:

$$\mathbb{E}\{\cos(\pi X)\} = \cos(0) \times p_0 + \cos(\pi) \times p_1 = p_0 - p_1$$

and finally:

$$\phi_X(u) = p_0 e^{ju\times 0} + p_1 e^{ju\times 1} = p_0 + p_1 e^{ju}$$

■

Definition 7.10 (n-th order moment)
The n-th order moment is the mathematical expectation of the function $f(x) = x^n$.

Definition 7.11 (Mean, variance)
The mean of the random variable X is defined as the first order moment, that is to say $\mathbb{E}\{X\}$. If the mean is equal to zero, the random variable is said to be centered. The variance of the random variable X is the quantity defined by:

$$\mathrm{var}(X) = \mathbb{E}\left\{(X - \mathbb{E}\{X\})^2\right\} = \mathbb{E}\left\{X^2\right\} - (\mathbb{E}\{X\})^2$$

The variance is always positive, and its square root is called the standard deviation.

The standard deviation can be interpreted as a measure of the random variable's fluctuations around its mean: the higher it is, the more the values of X are spread out around $\mathbb{E}\{X\}$.

Property 7.4 (Chebyshev inequality)
Let X be a random variable, with $\mathbb{E}\{X\}$ as its mean and $\mathrm{var}(X)$ as its variance. Then for any $\delta > 0$:

$$\begin{aligned} \Pr\left(\left|X - \mathbb{E}\{X\}\right| \geq \delta\right) &= \Pr\left(\mathbb{E}\{X\} - \delta \leq X \leq \mathbb{E}\{X\} + \delta\right) \\ &\leq \frac{\mathrm{var}(X)}{\delta^2} \end{aligned} \tag{7.13}$$

Inequality 7.13 means that the probability for X to deviate from its mean by $\pm\delta$ decreases when the variance decreases.

As an exercise, we are going to show that, for any constants a and b:

$$\mathbb{E}\{aX + b\} = a\mathbb{E}\{X\} + b \tag{7.14}$$

$$\mathrm{var}(aX + b) = a^2\mathrm{var}(X) \tag{7.15}$$

7.14 is a direct consequence of the integral's linearity. We assume that $Y = aX + b$, then $\mathrm{var}(Y) = \mathbb{E}\left\{(Y - \mathbb{E}\{Y\})^2\right\}$. By replacing $\mathbb{E}\{Y\} = a\mathbb{E}\{X\} + b$, we get $\mathrm{var}(Y) = \mathbb{E}\left\{a^2(X - \mathbb{E}\{X\})^2\right\} = a^2\mathrm{var}(X)$.

A generalization of these two results to random vectors (their components are random variables) will be given by property 7.6.

Definition 7.12 (Covariance, correlation)
Let (X, Y)[1] be two random variables. The covariance of X and Y is the quantity defined by:

$$\begin{aligned} \mathrm{cov}(X, Y) &= \mathbb{E}\{(X - \mathbb{E}\{X\})(Y^* - \mathbb{E}\{Y^*\})\} \\ &= \mathbb{E}\{XY^*\} - \mathbb{E}\{X\}\,\mathbb{E}\{Y^*\} \end{aligned} \tag{7.16}$$

X and Y are said to be uncorrelated if $\mathrm{cov}(X, Y) = 0$ that is to say if $\mathbb{E}\{XY^*\} = \mathbb{E}\{X\}\,\mathbb{E}\{Y^*\}$. The correlation coefficient is the quantity defined by:

$$\rho(X, Y) = \frac{\mathrm{cov}(X, Y)}{\sqrt{\mathrm{var}(X)}\sqrt{\mathrm{var}(Y)}} \tag{7.17}$$

Applying the Schwartz inequality gives us $-1 \leq \rho(X, Y) \leq 1$.

[1] Except in some particular cases, the random variables considered from now on will be real. However, the definitions involving the mean and the covariance can be generalized with no exceptions to complex variables by conjugating the second variable. This is indicated by a star ($*$) in the case of scalars and by the exponent H in the case of vectors.

Definition 7.13 (Mean vector and covariance matrix)
Let $\{X_1, \ldots, X_n\}$ be n random variables with the respective means $\mathbb{E}\{X_i\}$. The mean vector is the n dimension vector with the means $\mathbb{E}\{X_i\}$ as its components. The $n \times n$ covariance matrix $\mathbf{C}$ is the matrix with the generating element $C_{ij} = \operatorname{cov}(X_i, X_j)$ for $1 \leq i \leq n$ and $1 \leq j \leq n$.

Matrix notation: if we write

$$\mathbf{X} = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix}$$

to refer to the random vector with the random variable X_k as its k-th component, the mean-vector can be expressed:

$$\mathbb{E}\{\mathbf{X}\} = \begin{bmatrix} \mathbb{E}\{X_1\} \\ \vdots \\ \mathbb{E}\{X_n\} \end{bmatrix}$$

and the covariance matrix:

$$\mathbf{C} = \mathbb{E}\left\{(\mathbf{X} - \mathbb{E}\{\mathbf{X}\})(\mathbf{X} - \mathbb{E}\{\mathbf{X}\})^H\right\} = \mathbb{E}\left\{\mathbf{X}\mathbf{X}^H\right\} - \mathbb{E}\{\mathbf{X}\}\,\mathbb{E}\{\mathbf{X}\}^H \quad (7.18)$$

Notice that the diagonal elements of a covariance matrix represent the respective variances of the n random variables. They are therefore positive. *If the n random variables are uncorrelated, their covariance matrix is diagonal.*

Property 7.5 (Positivity of the covariance matrix)
Any covariance matrix is positive, meaning that for any vector $\mathbf{a}$, we have $\mathbf{a}^H\mathbf{C}\mathbf{a} \geq 0$.

To obtain this result, consider for any sequence of complex values $\{a_1, \cdots, a_N\}$ the random variable $Y = \sum_{k=1}^{N} a_k(X_k - \mathbb{E}\{X_k\})$. We have, of course, $\mathbb{E}\left\{|Y|^2\right\} \geq 0$ (because it is the mathematical expectation of a positive random variable). We will now express $\mathbb{E}\left\{|Y|^2\right\}$. We get:

$$\begin{aligned}
\mathbb{E}\left\{|Y|^2\right\} &= \mathbb{E}\left\{\sum_{k=1}^{N} a_k(X_k - \mathbb{E}\{X_k\}) \sum_{m=1}^{N} a_m^*(X_m - \mathbb{E}\{X_m\})^*\right\} \\
&= \sum_{k=1}^{N}\sum_{m=1}^{N} a_k \mathbb{E}\left\{(X_k - \mathbb{E}\{X_k\})(X_m - \mathbb{E}\{X_m\})^*\right\} a_m^* \\
&= \sum_{k=1}^{N}\sum_{m=1}^{N} a_k c_{k,m} a_m^* = \mathbf{a}^H\mathbf{C}\mathbf{a} \geq 0
\end{aligned}$$

Property 7.6 (Linear transformation of a random vector)
Let $\{X_1, \ldots, X_n\}$ be n random variables with $\mathbb{E}\{\mathbf{X}\}$ as their mean vector and $\mathbf{C}_X$ as their covariance matrix, and let $\{Y_1, \ldots, Y_q\}$ be q random variables obtained by the linear transformation:

$$\begin{bmatrix} Y_1 \\ \vdots \\ Y_q \end{bmatrix} = \mathbf{A} \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix} + \mathbf{b}$$

where $\mathbf{A}$ is a matrix and $\mathbf{b}$ is a non-random vector with the adequate sizes. We then have:

$$\begin{aligned} \mathbb{E}\{\mathbf{Y}\} &= \mathbf{A}\mathbb{E}\{\mathbf{X}\} + \mathbf{b} \\ \mathbf{C}_Y &= \mathbf{A}\mathbf{C}_X\mathbf{A}^H \end{aligned}$$

The first expression is a direct consequence of the integral's linearity. It leads to $\mathbf{Y} - \mathbb{E}\{\mathbf{Y}\} = \mathbf{A}(\mathbf{X} - \mathbb{E}\{\mathbf{X}\})$. We now lay down the second expression:

$$\begin{aligned} \mathbf{C}_Y &= \mathbb{E}\{(\mathbf{Y} - \mathbb{E}\{\mathbf{Y}\})(\mathbf{Y} - \mathbb{E}\{\mathbf{Y}\})^H\} \\ &= \mathbf{A}\mathbb{E}\{(\mathbf{X} - \mathbb{E}\{\mathbf{X}\})(\mathbf{X} - \mathbb{E}\{\mathbf{X}\})^H\}\mathbf{A}^H = \mathbf{A}\mathbf{C}_X\mathbf{A}^H \end{aligned}$$

Definition 7.14 (White sequence) *Let $\{X_1, \ldots, X_n\}$ be a set of n random variables. They are said to form a white sequence if $\mathrm{var}(X_i) = \sigma^2$ and if $\mathrm{cov}(X_i, X_j) = 0$ for $i \neq j$. Hence their covariance matrix can be expressed:*

$$\mathbf{C} = \sigma^2 \mathbf{I}_n$$

where $\mathbf{I}_n$ is the $n \times n$ identity matrix.

Property 7.7 (Independence ⇒ non-correlation)
The random variables $\{X_1, \ldots, X_n\}$ are independent, then uncorrelated, and hence their covariance matrix is diagonal. Usually the converse statement is false.

7.3 Common probability distributions

7.3.1 Uniform probability distribution on (a, b)

Definition 7.15 *A random variable X is said to be uniformly distributed on (a, b) with $b > a$ if its probability density function has the expression:*

$$p_X(x) = (b-a)^{-1}\mathbb{1}(x \in (a,b)) = \begin{cases} (b-a)^{-1} & \text{for } x \in (a,b) \\ 0 & \text{otherwise} \end{cases} \tag{7.19}$$

Notice that the set of all possible values of X is reduced to the interval (a, b), and that the probability of X belonging to an interval $(c, d) \subset (a, b)$ is equal to $(d-c)/(b-a)$, and is therefore proportional to the interval's length. Its probability density function is shown in Figure 7.1. It is constant in the interval (a, b).

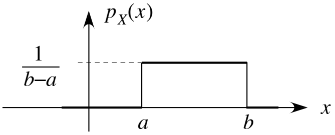


Figure 7.1 – *Probability density of the uniform probability distribution*

You can check as an exercise that its mean, its second-order moment and its variance are given respectively by:

$$\mathbb{E}\{X\} = \int_a^b \frac{x}{b-a}\,dx = \frac{a+b}{2}$$

$$\mathbb{E}\{X^2\} = \int_a^b \frac{x^2}{b-a}\,dx = \frac{a^2+ab+b^2}{3}$$

$$\operatorname{var}(X) = \mathbb{E}\{X^2\} - \mathbb{E}\{X\}^2 = \frac{(b-a)^2}{12}$$

An example of quantities that can be described by a uniformly distributed random variable is the errors that are made when, in calculations, numbers are rounded to D decimal places. When a large number of operations is performed, it can be assumed that the errors behave like random variables uniformly distributed between the values $-10^{-D}/2$ and $10^{-D}/2$. We will see on page 270 how this random variable model is used to describe uniform quantization noise.

7.3.2 Real Gaussian random variable

Definition 7.16 *A random variable X is said to be Gaussian, or normal, if all its values belong to $\mathbb{R}$ and if its characteristic function has the expression:*

$$\phi_X(u) = \exp\left(jmu - \frac{1}{2}\sigma^2 u^2\right)$$

where m is a real parameter and σ is a positive parameter. If $\sigma \neq 0$, it can be shown that the probability distribution has a probability density function with the expression:

$$p_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-m)^2}{2\sigma^2}\right) \tag{7.20}$$

We also check that its mean is equal to m and its variance to σ^2.

In physics, many phenomena, which are the combination of a multitude of microscopic effects, are distributed on the macroscopic scale according to a Gaussian probability distribution: this is the case of *background noise* in receptors. The Gaussian nature of these phenomena is a consequence of the *central limit theorem* [28]. Figure 7.2 shows the shape of the probability density function for the Gaussian random variable.

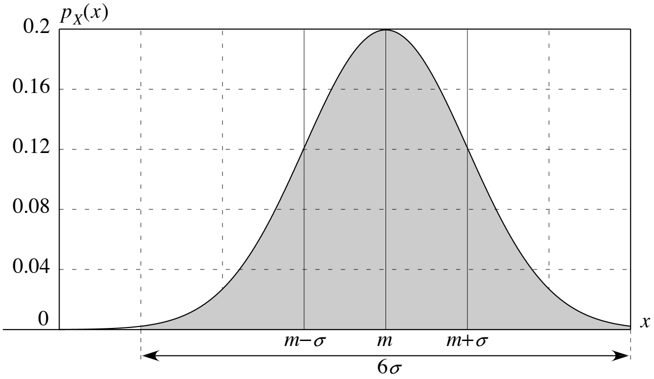


Figure 7.2 – *Probability density function of the Gaussian random variable and* 99% *confidence interval*

It can be checked numerically that more than 99% of the values belong to the interval $(m - 3\sigma, m + 3\sigma)$. The interval can then be called a 99% *confidence interval.* This leads us to a practical rule called the *3-sigma rule* for which the probability of "falling outside" this interval is less than 1%. If we restrict ourselves to a 95% confidence interval, we have to choose the interval $(m - 2\sigma, m + 2\sigma)$. If, on the contrary, we prefer a 99.9% confidence level, we have to take $(m - 4\sigma, m + 4\sigma)$.

7.3.3 Complex Gaussian random variable

In some problems, and particularly in the field of communications, the complex notation $X = U + jV$ is used, where U and V refer to two *real*, *Gaussian*, *centered*, *independent* random variables with the same variance $\sigma^2/2$. Because of independence (definition 7.7), the joint probability distribution of the pair (U, V) has the following probability density:

$$\begin{aligned} p_{UV}(u,v) &= p_U(u)p_V(v) = \frac{1}{\sigma\sqrt{\pi}}\exp\left(-\frac{u^2}{\sigma^2}\right) \times \frac{1}{\sigma\sqrt{\pi}}\exp\left(-\frac{v^2}{\sigma^2}\right) \\ &= \frac{1}{\pi\sigma^2}\exp\left(-\frac{u^2+v^2}{\sigma^2}\right) \end{aligned}$$

If we notice that $|x|^2 = u^2 + v^2$, and if we introduce the notation $p_X(x) = p_{UV}(u, v)$, we can also write:

$$p_X(x) = \frac{1}{\pi\sigma^2} \exp\left(-\frac{|x|^2}{\sigma^2}\right) \tag{7.21}$$

Expression 7.21 is called the probability density of a *complex Gaussian* random variable. The word *circular* is sometimes added as a reminder that the isodensity contours are the circles $u^2 + v^2 =$ constant.

Note that:

$$\begin{aligned} \mathbb{E}\{|X|^2\} &= \mathbb{E}\{XX^*\} = \mathbb{E}\{(U + jV)(U - jV)\} \\ &= \mathbb{E}\{U^2\} + \mathbb{E}\{V^2\} = \sigma^2 \end{aligned}$$

Expression 7.21 deserves a few words of warnings:

- the argument x is complex;
- if you compare expressions 7.21 and 7.20 of the probability density of a real Gaussian random variable, you will notice the disappearance of the factors 2 and of the square root in σ^2.

7.3.4 Generating the common probability distributions

MATLAB® has two random number generators:

1. the first one generates probability distributions *uniformly distributed* on $(0, 1)$ (see definition 7.15). This generator is called by the `rand` command;
2. the second is a centered *Gaussian* probability distribution (see definition 7.16), with a variance equal to 1. This generator is called by the `randn` command.

We will assume that when we use these generators several times, the resulting samples correspond to *independent* random variables. This is why the array `randn(4,1000)` can be considered as an experiment with a trial length of 1,000 on 4 independent, centered, Gaussian random variables with a variance equal to 1.

COMMENT: prior to version 4 of MATLAB® the `rand('uniform')` or `rand('normal')` initializes the random generator either for a probability distribution uniformly-distributed on the interval $(0, 1)$ or for centered, Gaussian probability distribution, with a variance equal to 1. Running the command `rand(k,c)` would then construct a matrix with `k` lines and `c` columns made up of random numbers of the corresponding probability distribution. The type of the currently used distributions could be known with the `rand('dist')` command. In the new versions of MATLAB®, the uniform generator is called with the `rand` command, and the Gaussian generator with the `randn` command. This is the only notation we will be using.

Example 7.2 (Uniform probability distribution on (a, b))
We wish to obtain a random variable X with a uniform probability distribution on (a, b), using the MATLAB® function `rand` which returns a random variable with a uniform probability distribution on $(0, 1)$. As an exercise, you can show that the random variable $X = (b - a)U + a$, where U is a uniform variable on $(0, 1)$, is uniform on (a, b). The `unifab.m` function given below generates N values uniformly distributed on (a, b):

```
function X=unifab(a,b,N)
%%===================================================%
%% Generating a r.v. uniformly distributed on (a,b) %
%% SYNOPSIS: X=UNIFAB(a,b,N)                         %
%%     a,b = Interval                                %
%%     N   = Number of samples                       %
%%     X   = Sequence of samples                     %
%%===================================================%
U=rand(1,N);
X=(b-a)*U+a;
return
```

In order to obtain 1,000 sample values of a random variable, uniform on $(-\pi, +\pi)$, type: `x=unifab(-pi,pi,1000);`. A sample of this type is represented in Figure 7.3.

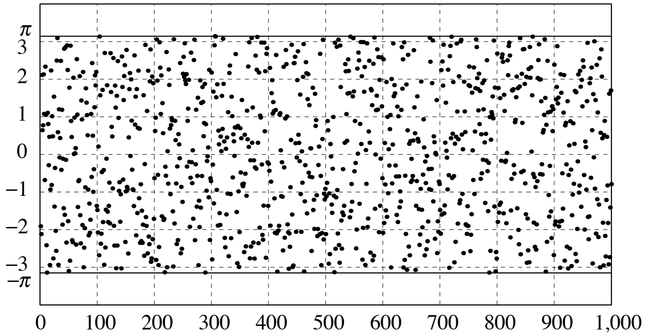


Figure 7.3 – *1,000 trials of a random variable uniformly distributed on* $[-\pi, +\pi]$

Now type `hist(X)`. The result is a diagram called a histogram, representing an estimation of the probability density shown in Figure 7.1.

Example 7.3 (Gaussian variable (m, σ^2))
Let Y be a centered Gaussian random variable with a variance of 1. If we apply relations 7.14 and 7.15, we can easily verify that the mean of the random variable $X = \sigma Y + m$ is m and that its variance is σ^2. We will see on page

263 that a direct consequence of the general definition of a Gaussian vector is that its Gaussian nature is unchanged by linear transformation. To get a 5,000 value sample of a Gaussian probability distribution with a mean equal to 4 and a variance equal to 7, type:

```
%===== HISTOG1D.M
clear; N=5000; m=4; sigma2=7;
X=sqrt(sigma2)*randn(1,N)+m;
figure(1); plot(X,'.')
lk=0.5; [nn,xx]=hist(X,(-7+m:lk:7+m));
pxchap=nn/(N*lk); figure(2); bar(xx,pxchap)
CG=1/sqrt(2*pi*sigma2); px=CG*exp(-(xx-m) .^2/(2*sigma2));
hold on; plot(xx,px,'o'); hold off; grid
```

Notice the use of formula 7.23 in the command `pxchap=nn/(N*lk)` to estimate the probability density using the results returned by the `hist` function. The trials are shown in Figure 7.4. The theoretical probability densities (∘) and the estimated ones (*bar chart*) are show in Figure 7.5.

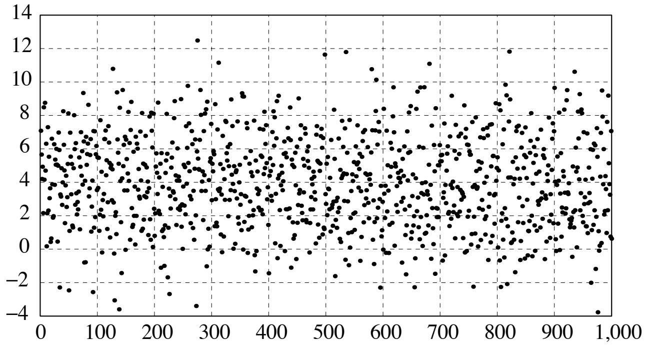


Figure 7.4 – *Gaussian probability distribution trials* $(m = 4, \sigma = 7)$

Example 7.4 (Complex Gaussian random variable)
To generate a 1,000 value sample of a centered, complex, Gaussian random variable (see expression 7.21), with a variance of 5, type the following program:

```
%===== GCOMPL.M
varX = 5 ;
U = sqrt(varX/2) * randn(1,1000) ;
V = sqrt(varX/2) * randn(1,1000) ;
X = U + j * V ;
```

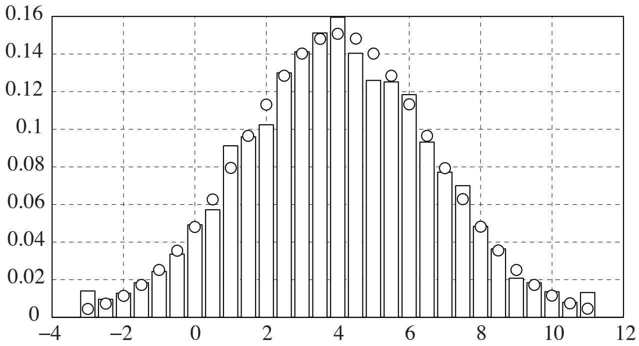


Figure 7.5 – *Histogram of the Gaussian probability distribution* $(m = 4, \sigma = 7)$

7.3.5 Estimating the probability density

A theorem inaccurately called the "law of large numbers" states that the probability for a random variable X to belong to an interval Δ can be approximated, if N is large enough, in the following way:

- consider N independent random variables with the same probability distribution as X; an experiment is conducted, leading to the trials $x_1, \dots,$ x_N;
- the number n of values in these trials that belong to Δ is determined;
- the approximation used for $\Pr(X \in \Delta)$ is n/N.

The quantity n/N is called the *empirical frequency*. This result can be used to estimate the probability density of $p_X(x)$ at the point x of the random variable X which is assumed to be continuous. By definition, we have:

$$\Pr(X \in \Delta) = \int_\Delta p_X(u)du$$

If Δ is a closed interval chosen small enough around the point x, we have $p_X(u) \approx p_X(x)$ and the second member is approximately equal to $p_X(x) \times \ell$ where ℓ refers to the length of Δ. This leads us to a practical formula for estimating $p_X(x)$:

$$p_X(x) \approx \frac{n}{N\ell} \tag{7.22}$$

where n is the number of observed points inside the closed interval Δ. Faced with a sample of N values, the procedure for estimating the probability density in P points can be summed up as follows:

Steps:

1. *The interval containing the observed values is partitioned in P sub-intervals $I_1, \cdots, I_P$ with the respective lengths $\ell_1, \cdots, \ell_P$, located around the points $x_1, \ldots, x_P$. Usually, the sub-intervals are chosen so that they all have the same length, and so that x_k is placed in the middle of the sub-interval I_k, except possibly for the first and last intervals.*
2. *The probability density at the point x_k is estimated by:*

$$\widehat{p}_X(x_k) = \frac{n_k}{N\ell_k} \tag{7.23}$$

where n_k is the number of points in the interval I_k. Note that $\sum_k \widehat{p}_X(x_k)\ell_k = 1$.

The choice of the value of P is a complex problem. What we can say is that P has to be large enough for the probability density to be properly estimated but also small enough for the number of points in each interval to remain large. $P = N^{1/3}$, for example, would be suitable, since it tends to infinity when N tends to infinity, and N/P also tends to infinity when N tends to infinity.

The MATLAB® function `hist` implements this procedure. The usual syntax is `[ndelta x0]=hist(X)`. In this case, `hist` automatically chooses ten values regularly spread out between the minimum and the maximum of the sample `X` with intervals of the same length.

The sequence `ndelta` returns the number of points of `X` placed around each element of the sequence `x0`. Thus a list of values for `x0` can be set as a parameter. Expression 7.23 is then used to find an estimation of the probability density.

7.3.6 Gaussian random vectors

Definition 7.17 (Gaussian vector) *$\{X_1, \ldots, X_n\}$ are said to be n jointly Gaussian variables, or that the length n vector $[X_1 \quad \ldots \quad X_n]^T$ is Gaussian, if any linear combination of its components, that is to say $Y = \mathbf{a}^T\mathbf{X}$ for any $\mathbf{a} = [a_1 \quad \ldots \quad a_n]^T$, is a Gaussian random variable.*

Theorem 7.2 (Probability distribution of a Gaussian vector) *It can be shown that the probability distribution of a length n Gaussian vector, with a length n mean vector $\mathbf{m}$ and an $(n \times n)$ covariance matrix has the characteristic function:*

$$\phi_{\mathbf{X}}(u_1, \ldots, u_n) = \exp\left(j\mathbf{m}^T\mathbf{u} - \frac{1}{2}\mathbf{u}^T\mathbf{C}\mathbf{u}\right) \tag{7.24}$$

where $\mathbf{u} = (u_1, \ldots, u_n)^T \in \mathbb{R}^n$. *Let* $\mathbf{x} = (x_1, \ldots, x_n)^T$. *If* $\det(\mathbf{C}) \neq 0$, *the probability distribution's density has the expression:*

$$p_{\mathbf{X}}(x_1, \ldots, x_n) = \frac{1}{(2\pi)^{n/2}\sqrt{\det(\mathbf{C})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})\right) \quad (7.25)$$

Theorem 7.3 (Gaussian case: non-correlation ⇒ independence)
If n *jointly Gaussian variables are uncorrelated, then they are independent.*

This is because if we replace $\mathbf{C} = \sigma^2\mathbf{I}$ in expression 7.25, $p_X(x_1, \ldots, x_n) = p_{X_1}(x_1) \ldots p_{X_n}(x_n)$, hence, according to 7.10, the variables are independent.

Theorem 7.4 (Linear transformation of a Gaussian vector)
Let $[X_1 \quad \ldots \quad X_n]^T$ *be a Gaussian vector with a mean vector* $\mathbf{m}_X$ *and a covariance matrix* $\mathbf{C}_X$. *The random vector* $\mathbf{Y} = \mathbf{AX} + \mathbf{b}$, *where* $\mathbf{A}$ *and* $\mathbf{b}$ *are a matrix and a vector respectively, with the ad hoc length, is Gaussian and we have:*

$$\mathbf{m}_Y = \mathbf{A}\mathbf{m}_X + \mathbf{b} \text{ and } \mathbf{C}_Y = \mathbf{A}\mathbf{C}_X\mathbf{A}^T$$

In other words, the Gaussian nature of a vector is untouched by linear transformations.

This result is a consequence of definition 7.17 and of property 7.6.

Exercise 7.1 (Confidence ellipse)

1. $p_{\mathbf{X}}(x_1, x_2)$ denotes the probability density of a length 2 random Gaussian vector, with the mean $\mathbf{m}$ and the covariance matrix $\mathbf{C}$, and let:

   $$\alpha = \Pr\left(\mathbf{X} \in \Delta(s)\right) = \int_{\Delta(s)} p_{\mathbf{X}}(x_1, x_2) dx_1 dx_2$$

   where $\Delta(s)$ is the set of points in the plane defined by:

   $$\Delta(s) = \left\{\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2 \ : \ (\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m}) < s\right\}$$

   the borderline of which in $\mathbb{R}^2$ is the ellipse centered on $\mathbf{m}$ with the equation $(\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m}) = s$.

   We will now determine the relation between s and α. The ellipse $\mathcal{E}$ is called the $100\alpha\%$ confidence ellipse of the variable $\mathbf{X}$.

   By making the variable change $\mathbf{Y} = \mathbf{C}^{-1/2}(\mathbf{X}-\mathbf{m})$, show that $s = -2\log(1-\alpha)$.

2. We assume $\mathbf{m} = [0 \quad 0]^T$, $\mathbf{C} = \begin{bmatrix} 2.3659 & -0.3787 \\ -0.3787 & 0.6427 \end{bmatrix}$ and $\alpha = 0.95$.

   Write a program:

   - that generates a length 2 Gaussian sample of $N = 200$ values, with a mean $\mathbf{m}$ and a covariance matrix $\mathbf{C}$, using a centered, white sample obtained with `y=randn(2,N)`;
   - that displays the points with the plane coordinates x, as well as the ellipse with the equation $(\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m}) = s$ (use the `ellipse` function, given on page 38 where $s = -2\log(1-\alpha)$);
   - that counts the number of points outside the ellipse and compares it to the value $(1-\alpha)N$. Think of using the `find` function for the condition $y_1^2 + y_2^2 > s$.

7.4 Generating an r.v. with any type of p.d.

Since the only generators MATLAB® provides are `rand` and `randn`, you may wonder whether it is possible to infer the function that can generate random variables with any type of probability distributions. The answer is yes, and one solution is given by the inversion of the cumulative distribution function [21].

Generating a discrete random variable

Let X be a discrete random variable, a sample of which we wish to generate. Let $\{a_0, a_1, \ldots, a_n \ldots, \}$ be the set of its values, $p_X(n) = \Pr(X = a_n)$ its probability distribution and $F_X(x) = \Pr(X \le x)$ its cumulative distribution function. Figure 7.6 shows the graph shape of the function $F_X(x)$. Its value in $x = a_k$ is expressed $F_X(a_k) = \sum_{n=0}^{k} p_X(n)$.

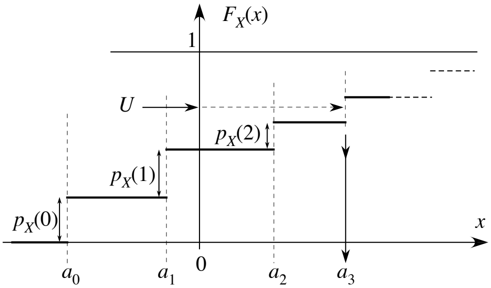


Figure 7.6 – *Cumulative distribution function of a discrete random variable*

Now consider a random variable U, uniformly-distributed on $(0,1)$, and let Y be the random variable obtained from U by inversion of the cumulative distribution function, which we write:

$$\text{If } U \in \left\{ \begin{array}{lll} [0, p_X(0)[& \text{then} & Y = a_0 \\ [p_X(0), p_X(0)+p_X(1)[& \text{then} & Y = a_1 \\ & \vdots & \\ [F_X(a_{k-1}), F_X(a_k)[& \text{then} & Y = a_k \\ & \vdots & \end{array} \right. \tag{7.26}$$

We will now show that the probability distribution of the obtained random variable Y is the very variable X we were looking for. Indeed, if we use the fact that U is uniform, we can successively write:

$$\begin{aligned} \Pr(Y = a_k) &= \Pr(U \in [F_X(a_{k-1}), F_X(a_k[) = \int_{F_X(a_{k-1})}^{F_X(a_k)} du \\ &= F_X(a_k) - F_X(a_{k-1}) = p_X(a_k) \end{aligned}$$

The variable Y constructed with procedure 7.26 obeys the expected probability distribution.

Example 7.5 (Uniform discrete random variable)
Generate a set of N values obeying a uniform discrete probability distribution on $\{0, \ldots, N-1\}$, meaning that $\Pr(X = k) = 1/N$ for $0 \leq k \leq N-1$. Draw the histogram of its values.

HINT: because the cumulative distribution function is such that $F_X(k) = k/N$ for $k \in \{0, \ldots, N-1\}$, expression 7.26 provides us with, if $U \in [k/N, (k+1)/N[$, the value $X = k/N$, meaning that X is simply the integer part of NU. Type:

```
%===== HISTOUNIF.M
clear; clf; nbp=3000; N=10; U=rand(1,nbp); X=ceil(N*U);
px=hist(X,(1:N)); bar(px/nbp); grid
```

As we expected, the obtained graph matches the uniform probability distribution $\Pr(X = k) = 1/10$. ■

Exercise 7.2 (Poisson distribution)
The random variable X, with possible values in $\mathbb{N}$, has a Poisson distribution when:

$$\Pr(X = k) = p_X(k) = \frac{a^k}{k!} e^{-a} \tag{7.27}$$

where a refers to a positive quantity called the *distribution parameter.*

1. Determine the mean and the variance of X.
2. Determine, for $k \in \mathbb{N}$, the recurrence relation that gives $p_X(k)$ as a function of $p_X(k-1)$, as well as the one that gives $F_X(k) = \Pr(X \leq k)$.
3. Using 7.26, write a program that generates a Poisson random variable with a parameter $a = 5$, using a random variable U uniformly-distributed on $(0, 1)$.
4. Using the `hist` function, check the result.

Theorem 7.5 (Variable change formula)
Let $x = f(u)$ be a bijective and differentiable function, and let U be a random variable, with the probability density $p_U(u)$. Then the random variable $X = f(U)$ has the following probability density:

$$p_X(x) = p_U(u)\left|\frac{du}{dx}\right| = \frac{p_U(u)}{\left|\frac{dx}{du}\right|} = \frac{p_U(g(x))}{|f'(g(x))|} \tag{7.28}$$

where $f'(u) = dx/du$ refers to the derivative of $f(u)$ and where $u = g(x)$ refers to the inverse function of $x = f(u)$, that is to say such that $g(f(u)) = u$.

Generating a continuous random variable

We now apply 7.28 in the particular case where U is a random variable uniformly distributed on $(0, 1)$ the probability density of which has the expression $p_U(u) = \mathbb{1}(u \in (0, 1))$. The probability distribution for the random variable $X = f(U)$ then has the probability density:

$$p_X(x) = \left|\frac{du}{dx}\right| \mathbb{1}\{g(x) \in (0, 1)\} \tag{7.29}$$

where $u = g(x)$ represents the inverse of $x = f(u)$. For $g(x)$, we will choose the function $F(x)$, where $F(x)$ is precisely the cumulative distribution function of the random variable we want to generate a sample of. We have $\frac{du}{dx} = F'(x) \geq 0$. By replacing it in 7.29 and by noticing that $F(x) \in [0, 1]$, we get $p_X(x) = F'(x)$, meaning that the probability distribution of X has the probability density $F'(x)$, which is the probability density of the expected distribution.

We can use this result to our advantage, to generate, using the uniform generator on $(0, 1)$, a sample of the random variable for a given probability density $p_X(x)$. Here is the algorithm:

Steps*:*

1. Determine the function $u = \int_{-\infty}^{x} p_X(t)dt$.

2. *Determine its inverse $x = g(u)$.*
3. *If U is a sample uniformly distributed on $(0, 1)$, then $X = g(U)$ is a sample whose distribution has the probability distribution $p_X(x)$.*

Example 7.6 (Exponential distribution)
A random variable has an exponential distribution if its values belong to $\mathbb{R}^+$ and its probability density has the expression:

$$p_X(x) = \lambda \exp(-\lambda x)\mathbb{1}(x \in [0, +\infty[) \tag{7.30}$$

where the parameter $\lambda \geq 0$.

1. Determine the mean and the variance of X.
2. Using the cumulative probability distribution of X, determine a function $X = g(U)$ such that X has an exponential distribution with the parameter λ when U has a uniform probability distribution on $(0, 1)$.
3. Check the result using the `hist` function.

HINT:

1. An integration by parts leads to $\mathbb{E}\{X\} = \lambda \int_0^{+\infty} x e^{-\lambda x} dx = 1/\lambda$. Likewise, $\mathbb{E}\{X^2\} = 2/\lambda^2 \Rightarrow \text{var}(X) = 1/\lambda^2$.
2. The cumulative distribution function of X has the expression $u = \int_0^x \lambda e^{-\lambda t} dt = 1 - e^{-\lambda x}$. If we “inverse” it, we get $x = -\log(1 - u)/\lambda$. Hence the variable $X = -\log(1 - U)/\lambda$ has an exponential distribution if U has a uniform probability distribution on $(0, 1)$. Because U and $(1 - U)$ have the same distribution, we can also simply choose $X = -\log(U)/\lambda$.
3. Type:

```
%===== EXPLAW.M
clear; N=3000; lambda=2; U=rand(N,1); X=-log(U)/lambda;
moyX=1/lambda; lk=moyX/10; maxx=max(X); pointsx=(0:lk:maxx);
[nn,xx]=hist(X,pointsx); bar(xx,nn/(N*lk)); hold on
%===== Theoretical exponential distribution
pth=lambda*exp(-lambda*pointsx); plot(pointsx,pth,'r')
hold off; grid
```

In the program, the step `lk`, used to estimate the probability density, is determined from the mean. ■

Exercise 7.3 (Rayleigh distribution)
X has a *Rayleigh* distribution if its probability density has the following expression:

$$p_X(x) = \frac{x}{\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right) \mathbb{1}(x \in [0, +\infty[) \tag{7.31}$$

Check that $\mathbb{E}\{X\} = \sigma\sqrt{\pi/2}$. Knowing that U has a uniform probability distribution on $(0, 1)$:

1. determine the function $X = g(U)$ such that X has a Rayleigh distribution;

2. use the `hist` to check the result.

Exercise 7.4 (Bernoulli distribution)
B is said to have a *Bernoulli* distribution (the kind of distribution you get if you flip a coin several times in a row) with a parameter p, if B is a random variable with only two possible values, 0 and 1, with the probabilities $\Pr(X = 1) = p$ and $\Pr(X = 0) = 1 - p$ respectively. Considering N independent Bernoulli variables B_n, let us assume:

$$S = \frac{1}{N}\sum_{n=1}^{N} B_n$$

1. Determine, as a function of p, the mean and the variance of B_n.

2. Determine, as a function of p, $\mathbb{E}\{B_k B_n\}$ for $k \neq n$. Remember that if random variables U and V are independent, they are uncorrelated, meaning that $\mathbb{E}\{UV\} = \mathbb{E}\{U\}\mathbb{E}\{V\}$.

3. Determine as a function of p and N the mean m and the variance σ^2 of the random variable S.

4. We assume that if N is large enough, the probability for S to be located in the interval $(m - 2\sigma, m + 2\sigma)$ is greater than 95%. This means we can say that S provides an estimation of m with the relative precision $\varepsilon_r = 2\sigma/m$. Determine the expression of ε_r as a function of p and of N. Use this result to show that for $\varepsilon_r = 10\%$, and for small values of p, an approximate value of N is given by $N \approx 400/p$.

5. Write a program that calculates the length N of a Bernoulli sequence with the parameter $p = 0.1$ so that the empirical mean S is an approximation of p with an accuracy of ε_r.

The Bernoulli distribution can be used, among other applications, as a random sequence of bits used to simulate a digital communications system, or as it is explained in example 7.7, as a model for describing errors.

Example 7.7 (Error probability estimation)
Consider a random experiment where a sequence of values is received with an error probability p. We are going to try and estimate p using a sequence of N observations. In order to do this, a given sequence of length N is sent and compared to the received sequence. The question we are faced with is *"what must be the value of N to estimate the error probability with an accuracy of* 10%*?"*.

The model used for describing the error sequence is a sequence of random variables B_n such that $B_n = 0$ if the values in the n-th position are identical in the original sequence and the erroneous sequence and $B_n = 1$ if they are different. This way, the random variable:

$$S = \frac{1}{N} \sum_{n=1}^{N} B_n \tag{7.32}$$

gives an estimation of the probability error p. If we assume that the random variables B_n are independent, we can use the results obtained in exercise 7.4. To estimate p with an accuracy ε_r roughly equal to 10% and a 95% confidence interval, we need to start with a sequence with a length of $N \approx 400/p$. If we restrict ourselves to a 70% confidence interval, the same accuracy $\varepsilon_r = 0.1$ is achieved for $N \approx 100/p$. This value is often the one used in practice to calculate the length of a test. Note that Np represents approximately the number of errors. This leads us to adopting the following rule:

To estimate an error probability with a precision of 10% *and a confidence level of* 70%*, you need to see a hundred errors "go by". Hence the estimation must be performed on a sequence with a length roughly equal to* $100/p$.

If the order of magnitude wanted for p is 10^{-5}, you need $N = 10{,}000{,}000$. Such a length can require a long simulation time, even with a fast computer. This is a very common problem in the field of digital communications (see exercise 12.25).

7.5 Uniform quantization

Quantization provides a non-trivial example using the concepts explained in this chapter. The practical implications of the results are fundamental. In this paragraph, we will only discuss the very simple case of the *uniform scalar* quantization.

The uniform quantization operation on N bits consists of dividing the interval $(-A, +A)$ in 2^N sub-intervals of the same length $q = 2A/2^N$. q is called the *quantization step*. When the quantization operation is performed, each sample X is associated with the N-bit coded number of the interval it belongs to. When the signal is reconstructed, this number is replaced by the median value of the interval. If X denotes the sample we want to quantize and Y denotes the reconstructed value, we have:

$$Y = kq + q/2 \quad \text{when} \quad kq \leq X < (k+1)q$$

Usually ADCs use a two's complement binary coding for the quantized samples. The code takes values between -2^{N-1} and $+2^{N-1} - 1$ where N is the number of bits used for coding. The conversion law can be described by Figure 7.7.

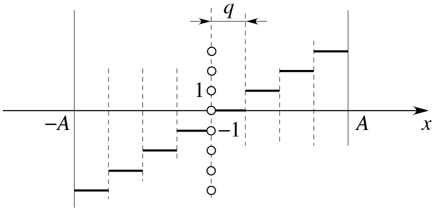


Figure 7.7 – *Quantization and two's complement binary coding with* 3 *bits*

Therefore, there is a gap between the "true" value and the reconstructed value. If $\varepsilon = X - Y$ refers to this gap, called the *quantization noise*, we can write $X = Y + \varepsilon$. The quantization operation is the equivalent of adding a noise with the power $\mathbb{E}\{\varepsilon^2\}$.

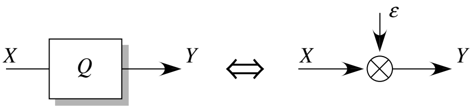


Figure 7.8 – *Uniform quantization*

Although it is possible to determine the probability distribution of ε using that of X, it is often sufficient to just assume that ε is a uniform random variable on $(-q/2, +q/2)$. This means that $\mathbb{E}\{\varepsilon\} = 0$ and that:

$$\mathbb{E}\{\varepsilon^2\} = \int_{-q/2}^{+q/2} \varepsilon^2 \frac{1}{q} d\varepsilon = \frac{q^2}{12} \tag{7.33}$$

The uniformity hypothesis implies the absence of *clipping*, meaning that none of the values we wish to quantize are located outside the range $(-A, +A)$. Otherwise, ε can assume much greater values than $q/2$. It just needs to be made sure that the amplitudes of X higher than A have a negligible probability. In the case where X is a centered random variable with a variance σ^2, we usually choose:

$$A = F_c \sigma$$

where F_c is called *Clipping Factor* (*CF*). Thus, if X is Gaussian, by adopting the "3 *sigma*" rule (page 258), corresponding to a 99% confidence level, $F_c = 3$. For a speech signal, the Gaussian hypothesis usually works poorly, and the value of F_c is rather chosen roughly equal to 4.

Exercise 7.5 (Signal-to-quantization noise ratio)
Consider the uniform quantization of an observation X described with a centered random variable with a variance σ^2. We wish to find the expression for the level of noise quantization as a function of the number of bits used for coding.

1. The signal-to-quantization noise ratio (SNR) is defined by:

   $$\mathrm{SNR} = 10 \log_{10} \left(\frac{\mathbb{E}\{X^2\}}{\mathbb{E}\{\varepsilon^2\}} \right)$$

   Show that in the absence of clipping, that is if $A = F_c \sigma$ with F_c high enough, the SNR's expression as a function of the number N of bits used for coding the samples is:

   $$\mathrm{SNR} = 10 \log_{10}(\mathbb{E}\{X^2\}/\mathbb{E}\{\varepsilon^2\}) = 6N + 10 \log_{10}(3/F_c^2) \qquad (7.34)$$

2. Write a function performing the two's complement binary coding of a sequence.

3. Write a program to check the hypothesis of uniform quantization noise distribution. Perform a simulation to measure the SNR for values of N from 1 to 7.

What should be remembered from formula 7.34 of exercise 7.5 is the following rule, called the 6 *dB per bit rule*:

In uniform quantization, the signal-to-quantization noise ratio is enhanced by 6 *dB every time* 1 *bit is added to the quantizer.*

Chapter 8

Random Processes

8.1 Introduction

At the beginning of Chapter 7, we pointed out that the concept of random variable was needed to describe with a model the variability of certain phenomena said to be random. Speech signal observed at a microphone's output is an example. There is no use to try and describe it with a deterministic expression such as $x(t) = A\cos(2\pi f_0 t)$, which is relevant however when describing electrical voltage, hence the idea of using random variables for describing the phenomenon at every instant. This leads us to the following definition.

Definition 8.1 *A random process is a set of time-indexed random variables $X(t)$ defined in the same probability space. If the possible values for t belong to $\mathbb{R}$, the process is called a continuous-time random process. If the possible values for t belong to $\mathbb{Z}$, then we are dealing with a discrete-time random process*[1].

The definition implies that a random process associates a real value called a realization with every *instant* t and every *outcome* ω. A *random process* can therefore be interpreted as two different perspectives (Figure 8.1):

1. either as a set of functions of time, also called *trajectories*, each one associated with an outcome;

2. or as a set of *random variables*, each one associated with a given time.

In MATLAB®, the `randn` function makes it possible to simulate the trajectories of a zero-mean gaussian random process with a variance of 1. In the following program, `x` is a matrix with 4 columns and 100 lines:

```
%===== TRAJ1.M
x=randn(100,4);
for k=1:4
```

[1] We will often use n, k, ℓ, m, ... to denote time for a discrete-time random process.

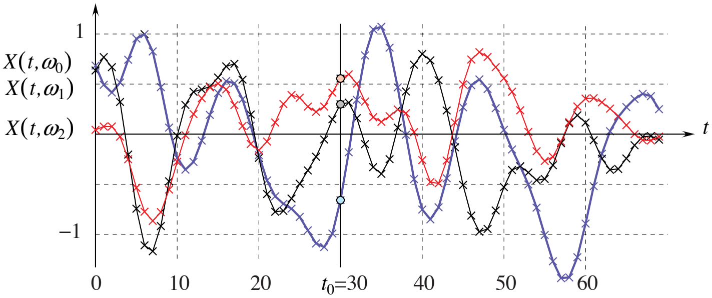


Figure 8.1 – *Trajectories of a random process*

```
    subplot(2,2,k); plot(x(:,k)); grid
end
```

x can be considered as the representation of four trajectories for the same random process, for an observation time of 100 points (Figure 8.2).

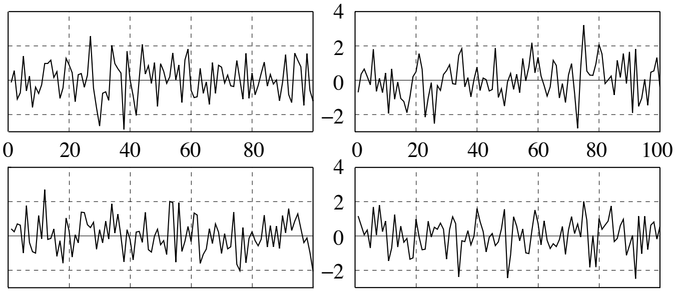


Figure 8.2 – *Examples of random process trajectories*

8.2 Wide-sense stationary processes

In some processings involving random processes, particularly in linear filtering, the signals are assumed to be *stationary*, and only the first and second order moments are taken into account. The two trajectories represented in Figure 8.3 illustrate these concepts. We might say, after observing them, that:

– the behavior of the signals with time show a certain permanence. Their properties do not depend on the time origin;

– the mean seems to be equal to zero;

- most of the power is located around roughly 110 Hz, because 11 oscillations are observed over a duration of 0.1 s, hence about 110 oscillations per second.

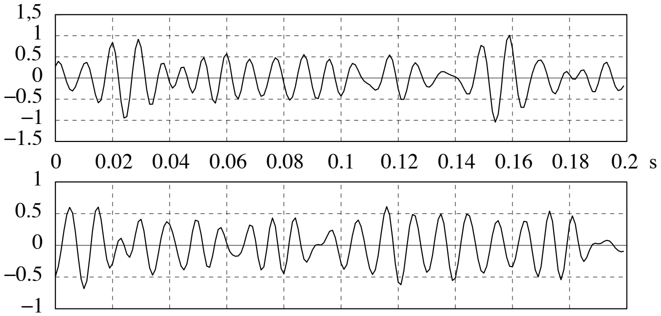


Figure 8.3 – *The two representations give an idea of the meaning of the word "stationary"*

You will find here the mathematical definitions associated with these concepts for random processes, and such a process is referred to as a *Wide sense stationary random process*, or WSS.

8.2.1 Definitions and properties of WSS processes

Definition 8.2 (Mean) *The mean of a random process is the mathematical expectation of the random variable $X(t)$. This quantity, which depends on the time t, is a deterministic function of t, which will be denoted by:*

$$m_X(t) = \mathbb{E}\{X(t)\} \tag{8.1}$$

Definition 8.3 (Autocovariance function) *Let $X(t)$ be a random process. The autocovariance function is the function of t_1 and t_2 defined by:*

$$R_{XX}(t_1, t_2) = \mathbb{E}\{X_c(t_1)X_c^*(t_2)\} \tag{8.2}$$

where $X_c(t) = X(t) - \mathbb{E}\{X(t)\}$ refers to the centered process.

Definition 8.4 (Autocorrelation function) *Let $X(t)$ be a random process. The* autocorrelation function *is the function of t_1 and t_2 defined by:*

$$\rho_{XX}(t_1, t_2) = \frac{\mathbb{E}\{X_c(t_1)X_c^*(t_2)\}}{\sqrt{\mathbb{E}\{|X_c(t_1)|^2\}\,\mathbb{E}\{|X_c(t_2)|^2\}}} \tag{8.3}$$

where $X_c(t) = X(t) - \mathbb{E}\{X(t)\}$ refers to the centered process. *The Schwartz inequality tells us that for any random variables X_1 and X_2:*

$$|\mathbb{E}\{X_1X_2^*\}|^2 \leq \mathbb{E}\{\,|X_1|^2\}\,\mathbb{E}\{\,|X_2|^2\} \tag{8.4}$$

Inequality 8.4 implies that:

$$\forall t_1, t_2,\ |\rho_{XX}(t_1, t_2)| \leq 1 \tag{8.5}$$

A simple calculation shows that:

$$R_{XX}(t_1, t_2) = \mathbb{E}\{X(t_1)X^*(t_2)\} - m_X(t_1)m_X^*(t_2) \tag{8.6}$$

Definition 8.5 (Covariance function) *The covariance function of two distinct processes $X(t)$ and $Y(t)$ is defined by:*

$$R_{XY}(t_1, t_2) = \mathbb{E}\{X_c(t_1)Y_c^*(t_2)\} = \mathbb{E}\{X(t_1)Y^*(t_2)\} - m_X(t_1)m_Y^*(t_2) \tag{8.7}$$

Of course, in the case of a real process, it is not useful to have the conjugation $(^*)$ appear in expressions 8.6 and 8.7. The autocovariance function is always a deterministic function of t_1 and t_2. From now on, if there is no possibility of confusion, we will omit the index $_{XX}$ when writing something of the type $R_{XX}(t_1, t_2)$.

In the general case, the autocovariance function $R_{XX}(t_1, t_2)$ depends *separately* on t_1 and t_2. In the particular case where it only depends on $t_1 - t_2$, the time origin does not determine the level of covariance. This implies that the trajectories of the process have an almost eternal permanence. The term *stationary* is associated with this property. This concept is absolutely fundamental in signal processing. It leads to the following definition.

Definition 8.6 (Second order stationary random process) *A random process is said to be "wide sense second order stationary" (WSS), or simply "second order stationary", if it obeys the following properties[2]:*

- *the mean $\mathbb{E}\{X(t)\} = m$ is independent of t;*
- $\mathbb{E}\{|X(t)|^2\} < +\infty$*;*
- *the autocovariance function $\mathbb{E}\{X_c(t_1)X_c^*(t_2)\} = R(\tau)$ depends on the time difference $\tau = t_1 - t_2$.*

We have, according to relation 8.7, $\mathbb{E}\{X(t_1)X^*(t_2)\} = \mathbb{E}\{X_c(t_1)X_c^*(t_2)\} + m_X(t_1)m_X^*(t_2)$. If we let $t_2 = t$ and $t_1 = t + \tau$ we get:

$$\mathbb{E}\{X(t+\tau)X^*(t)\} = R(\tau) + |m|^2$$

which depends only on τ.

[2] In the case of continuous-time processes, we add that the autocovariance function is continuous at the origin.

Example 8.1 (Complex harmonic process)
Consider the complex random process defined by:

$$X(t) = \sum_{k=1}^{P} \alpha_k e^{2j\pi f_k t} \tag{8.8}$$

where $\{f_k\}$ refers to a deterministic sequence of P frequencies and $\{\alpha_k\}$ to a sequence of P *zero-mean*, *uncorrelated* complex random variables with the respective variances σ_k^2. Such a process is called a *harmonic process*. It does not matter here whether t is an integer or a real number.

First, we calculate the mean of $X(t)$. Because the expectation of the sum is the sum of the expectations, we get $\mathbb{E}\{X(t)\} = 0$. Hence the process is zero-mean. Its autocovariance function has the expression:

$$\mathbb{E}\{X(t+\tau)X^*(t)\} = \sum_{k=1}^{P}\sum_{n=1}^{P} \mathbb{E}\{\alpha_k\alpha_n^*\}\, e^{2j\pi f_k(t+\tau)} e^{-2j\pi f_n t}$$

Because by hypothesis, $\mathbb{E}\{|\alpha_k|^2\} = \sigma_k^2$ and $\mathbb{E}\{\alpha_k\alpha_n^*\} = 0$ for $k \neq n$, we get:

$$\mathbb{E}\{X(t+\tau)X^*(t)\} = \sum_{k=1}^{P} \sigma_k^2 e^{2j\pi f_k \tau} \tag{8.9}$$

$\mathbb{E}\{X(t+\tau)X^*(t)\}$ depends only on τ. This process is therefore WSS.

Example 8.2 (Real harmonic process)
Consider the random process:

$$X(t) = \sum_{k=1}^{P} A_k \cos(2\pi f_k t + \Phi_k) \tag{8.10}$$

where $\{f_k\}$ refers to a deterministic sequence of P frequencies, $\{A_k\}$ to a sequence of P independent, zero-mean, real random variables, with the respective variances σ_k^2 and $\{\Phi_k\}$ to a sequence of P uniform random variables on $(0,2\pi)$, independent of one another and of A_k. A calculation similar to the previous one leads to $\mathbb{E}\{X(t)\} = 0$. We now calculate the autocovariance function. We get:

$$\mathbb{E}\{X(t+\tau)X^*(t)\} = \sum_{k=1}^{P}\sum_{n=1}^{P} \mathbb{E}\{A_kA_n\cos(2\pi f_k(t+\tau)+\Phi_k)\cos(2\pi f_n t+\Phi_n)\}$$

$$= \sum_{k=1}^{P}\sum_{n=1}^{P} \mathbb{E}\{A_kA_n\}\,\mathbb{E}\{\cos(2\pi f_k(t+\tau)+\Phi_k)\cos(2\pi f_n t+\Phi_n)\}$$

$$= \sum_{k=1}^{P} \mathbb{E}\{A_k^2\}\,\mathbb{E}\{\cos(2\pi f_k(t+\tau)+\Phi_k)\cos(2\pi f_k t+\Phi_k)\}$$

where we used the non correlation of the A_k, then the independence of A_k and of Φ_k. Next, we get:

$$\begin{aligned}\mathbb{E}\{X(t+\tau)X^*(t)\} &= \frac{1}{2}\sum_{k=1}^{P}\sigma_k^2\mathbb{E}\{\cos(2\pi f_k\tau)+\cos(2\pi f_k(2t+\tau)+2\Phi_k)\}\\ &= \frac{1}{2}\sum_{k=1}^{P}\sigma_k^2\left[\cos(2\pi f_k\tau)+\mathbb{E}\{\cos(2\pi f_k(2t+\tau)+2\Phi_k)\}\right]\\ &= \frac{1}{2}\sum_{k=1}^{P}\sigma_k^2\cos(2\pi f_k\tau) \qquad (8.11)\end{aligned}$$

where we used the hypothesis according to which Φ_k is uniform in $(0, 2\pi)$ and therefore that:

$$\mathbb{E}\{\cos(2\pi f_k(2t+\tau)+2\Phi_k)\} = \int_0^{2\pi}\cos(2\pi f_k(2t+\tau)+2\phi)\frac{1}{2\pi}\,d\phi = 0$$

As a conclusion, the autocovariance function of a real harmonic process depends only on τ and therefore the process is WSS.

COMMENTS: if the autocovariance function (expressions 8.9 and 8.11) of a real or complex harmonic process is the sum of periodic functions, then the variables $X(t+\tau)$ and $X(t)$ remain correlated, even for large time differences. This is called a memory effect, and it lasts indefinitely.

The property 11.3, which we will see further on, gives it a more precise mathematical meaning: a harmonic process is perfectly predictable from its last P values.

Unlike harmonic processes, there are WSS processes for which the autocovariance function $R(\tau)$ tends to 0 when τ tends to infinity. This can be interpreted as a *memory loss* of the process occurring with time.

8.2.2 Spectral representation of a WSS process

Definition 8.7 (Spectral density)
Let $X(t)$ be a WSS process with the autocovariance function $R(\tau)$. The Fourier transform of $R(\tau)$ is called the power spectral density (or PSD), or the spectrum. For continuous-time WSS random processes, the PSD therefore has the expression:

$$S(f) = \int_{-\infty}^{+\infty} R(\tau)e^{-2j\pi f\tau}d\tau \qquad (8.12)$$

and for discrete-time WSS random processes:

$$S(f) = \sum_{k=-\infty}^{+\infty} R(k)e^{-2j\pi fk} \qquad (8.13)$$

Power is defined as:

$$P = \mathbb{E}\{|X(t)|^2\} = R(0) + |m|^2 \tag{8.14}$$

The power's square root is also called the *root mean square.* We will often be dealing with zero-mean processes. In such cases, the power is equal to the autocovariance function's value at the origin.

By inversion of the FT, we have, for continuous-time WSS random processes:

$$R(\tau) = \int_{-\infty}^{+\infty} S(f) e^{2j\pi f\tau} df \text{ with } \tau \in \mathbb{R} \tag{8.15}$$

and for discrete-time WSS random processes:

$$R(k) = \int_{-1/2}^{+1/2} S(f) e^{2j\pi fk} df \text{ with } k \in \mathbb{Z} \tag{8.16}$$

Note that because of 8.15 and 8.16, $R(0)$ is merely the integral of $S(f)$.

The following result can be proven [14]:

Theorem 8.1 (Positivity of the PSD) *Let $X(t)$ be a WSS random process, and let $S(f)$ be its PSD. We have:*

$$S(f) \geq 0 \tag{8.17}$$

In the particular case of a real process, the PSD is an even function, that is $S(f) = S(-f)$.

It should be noted that the positive nature of $S(f)$ is directly related to the positive nature of the covariance, a property we showed on page 255.

Property 8.1 *Let $X(t)$ be a WSS random process. We have:*

1. *Hermitian symmetry: $R(\tau) = R^*(-\tau)$. Therefore, we only need to evaluate $R(\tau)$ for $\tau \geq 0$.*

2. *Positivity property: for any N, for* any sequence *of times $\{t_0, \ldots, t_{N-1}\}$, and for* any sequence *of complex values $\{a_0, \ldots, a_{N-1}\}$:*

$$\mathbf{a}^H \mathbf{R} \mathbf{a} = \sum_{k=0}^{N-1} \sum_{m=0}^{N-1} a_k a_m^* R(t_k - t_m) \geq 0$$

where $\mathbf{R}$ is the $N \times N$ covariance matrix constructed from the covariance function $R(\tau)$ of the process, where $\mathbf{a} = [a_0, \ldots, a_{N-1}]^T$ and where the exponent H indicates a transpose-conjugation (see also page 255).

3. *If the process has a mean different from zero, a "peak" with an amplitude $|m|^2$ is attached to the spectrum at the origin ($f = 0$). This peak at the origin, which simply indicates the presence of a non-zero mean, is called the* continuous component *of a process. Using the Dirac distribution amounts to choosing the Fourier transform of the second order moment as the definition of the spectrum. Indeed, we have:*

$$\mathbb{E}\{X(t+\tau)X^*(t)\} = R(\tau) + |m|^2$$

which has $S(f) + |m|^2\delta(f)$ as its Fourier transform.

4. *The complex and real harmonic processes, expressions 8.8 and 8.10, have periodic autocovariance functions, expressions 8.9 and 8.11. Therefore, it does not exactly have a Fourier transform. However, we can find a meaning to the PSD using Fourier series:*

   – *in the case of a complex process:*

$$S(f) = \sum_{k=1}^{P} \sigma_k^2 \delta(f + f_k) \tag{8.18}$$

   – *and in the case of a real process:*

$$S(f) = \frac{1}{4}\sum_{k=1}^{P} \sigma_k^2 \delta(f + f_k) + \frac{1}{4}\sum_{k=1}^{P} \sigma_k^2 \delta(f - f_k) \tag{8.19}$$

   The PSD comprises peaks that indicate the presence in the signal of sinusoidal components with uncorrelated amplitudes.

5. *As it was the case with the deterministic description, the spectrum represents the distribution (or localization) of the power along the frequency axis. The power is given by:*

$$\begin{cases} P = R(0) + |m|^2 = \displaystyle\int_{-\infty}^{+\infty} S(f)df + |m|^2 & \textit{(continuous-time)} \\ P = R(0) + |m|^2 = \displaystyle\int_{-1/2}^{+1/2} S(f)df + |m|^2 & \textit{(discrete-time)} \end{cases} \tag{8.20}$$

COMMENT: the fact that the PSD of an observed process contains peaks can be used in some synchronization systems to retrieve, using a very narrow band-pass filtering, a harmonic component with the same phase as a particular component of the observed process.

We admit without proof the following result: [14]

Property 8.2 (Characterization of positivity) *Consider a sequence $\Gamma(k)$ with $k \in \mathbb{Z}$ such that $\Gamma(k) = \Gamma^*(-k)$ and $\sum_k |\Gamma(k)| < +\infty$. This sequence is the covariance sequence of a WSS process if and only if, for all f:*

$$S(f) = \sum_{k=-\infty}^{+\infty} \Gamma(\tau) e^{-2j\pi f k} \geq 0$$

Example 8.3 Consider the real sequence $\Gamma(k) = 1 \times \mathbb{1}(k=0) + a \times \mathbb{1}(k = \pm 1)$. Determine the condition on a such that the sequence is the covariance sequence of a WSS process.

HINT: obviously we have, for any a, $\sum_k |\Gamma(k)| < +\infty$. Using property 8.2, $\Gamma(k)$ is a sequence of covariance if and only if:

$$S(f) = 1 + 2a\cos(2\pi f) \geq 0$$

The condition $S(f) \geq 0$ is equivalent to $|a| < 1/2$. Notice that a represents the correlation coefficient $\rho(1)$. Hence, after 8.5, we already knew that $|\rho(1)| \leq 1$. The fact that the sequence is a covariance sequence of a WSS process imposes a stronger condition. ■

Studying methods that make it possible to estimate the spectra of a second order stationary random process is an important field in signal processing. We will discuss this later on.

Positive Toeplitz matrix

Consider a WSS *discrete-time* random process $X(n)$. We are going to determine the covariance matrix at any K consecutive times of the process. If we start at the times $\{n, n+1, \ldots, n+K-1\}$[3], the $K \times K$ covariance matrix has the expression:

$$\begin{aligned}
\mathbf{R} &= \mathbb{E}\left\{ \begin{bmatrix} X_c(n) \\ X_c(n+1) \\ \vdots \\ X_c(n+K-1) \end{bmatrix} \begin{bmatrix} X_c^*(n) & X_c^*(n+1) & \ldots & X_c^*(n+K-1) \end{bmatrix} \right\} \\
&= \begin{bmatrix} R(0) & R(-1) & \ldots & R(-K+1) \\ R(1) & R(0) & R(-1) & \ldots \\ \vdots & \ddots & & R(-1) \\ R(K-1) & \ldots & R(1) & R(0) \end{bmatrix}
\end{aligned}$$

[3] Most of the time, we will write the time sequence from left to right by increasing times.

$$\mathbf{R} = \begin{bmatrix} R(0) & R^*(1) & \dots & R^*(K-1) \\ R(1) & R(0) & R^*(1) & \dots \\ \vdots & \ddots & & R^*(1) \\ R(K-1) & \dots & R(1) & R(0) \end{bmatrix}$$

Notice that $\mathbf{R} = \mathbf{R}^H$ and that because of the *stationarity* of the process, the matrix $\mathbf{R}$ is such that the lines parallel to the main diagonal are comprised of equal terms. This type of matrix is called a *Toeplitz matrix*.

The MATLAB® function `toeplitz(V)` allows you to construct, from the vector $\mathbf{V} = [V(0) \dots V(K-1)]^T$, the square *hermitian* Toeplitz matrix the *first line* of which is $[V(0), V(1), \dots, V(K-1)]$.

Definition 8.8 (Gaussian random process)
A random process $X(t)$ is Gaussian if, for any k, and for any time sequence $\{t_1, \dots, t_k\}$, the vector $[X(t_1), \dots, X(t_k)]$ is Gaussian.

For the definition of a Gaussian vector, see 7.17.

Definition 8.9 (White noise) *Discrete-time* white noise *is the name given to a WSS, zero-mean, random process $X(n)$, the covariance function of which can be written:*

$$R(k) = \mathbb{E}\{X(n+)X(n)\} = \begin{cases} R_0 & \textit{when} \quad k = 0 \\ 0 & \textit{when} \quad k \neq 0 \end{cases}$$

Because of formula 8.13, which provides us with the spectrum, the power spectral density is constant, and has the expression:

$$S(f) = R_0 \tag{8.21}$$

In the continuous-time case, a definition similar to 8.9 poses a problem, because it leads to a random process of infinite power (the integral $\int_{\mathbb{R}} S(f)df$ diverges) and the autocovariance function can only be defined in the distributions context. Thus, we have:

$$S(f) = R_0 \Leftrightarrow R(\tau) = R_0\delta(\tau)$$

where $\delta(\tau)$ now refers to the Dirac distribution. We must say however, that in most practical cases, the calculations performed with the Dirac distribution lead to results that coincide with those obtained by starting off with a B band noise (which does not lead to an infinite power problem), and then making B tend to infinity.

The word *white* comes from the analogy made with white light, for which the power is uniformly distributed among all the optical frequencies.

White noise is the archetype of models used in practice for describing noise. In communications systems, for example, it describes every kind of noise caused by thermal phenomena in the transmission chain, from the emitter to the receiver. The rounding and quantization noises that occur in a digital processing system are another example.

Although a process often is both Gaussian and white, particularly in thermal noise models, there is no implication between these two properties. Thus, a random process can be *white without being Gaussian* or *Gaussian without being white.*

Example 8.4 (Trajectory of a noisy sine)
Write a program that displays a sequence of 30 samples taken at the frequency $F_s = 1{,}000$ Hz, from a signal $X(t) = s(t) + B(t)$, sum of a sine $s(t)$ with a frequency of $F_0 = 80$ Hz and of a zero-mean, Gaussian, white noise $B(t)$. The power of $B(t)$ is chosen so as to have a signal-to-noise ratio equal to 15 dB, knowing that the signal $s(t)$ has an amplitude of 3.

HINT: having a signal-to-noise ratio equal to 15 dB means that the ratio r_P of the signal's power to the noise's power is such that $10\log_{10}(r_P) = 15$, and therefore $r_P = 10^{1.5}$. Because the power of a sine with an amplitude A is equal to $P = A^2/2$, the noise variance has to be $\sigma^2 = A^2/2r_P$. Since $A = 3$, this leads to $\sigma = 3/\sqrt{2 \times 10^{1.5}}$.

To obtain a trajectory, type:

```
%===== SIN80BR.M
N=30; F0=80; Fs=1000; tps=(0:N-1)/Fs;
sigma=3/sqrt(2*10 ^ 1.5);     %===== SNR
s=3*cos(2 * pi * F0 * tps);
x=s + sigma*randn(1,length(tps));
plot(tps,s,tps,x,'o'); grid
```

The result is shown in Figure 8.4. A cross indicates a sample without noise, and a circle indicates a noisy sample. ∎

Example 8.5 (Linear transformation of a WSS process)
Let $W(n)$ be a zero-mean, WSS, *discrete-time* random process. $R_{WW}(k)$ refers to the autocovariance function and $\mathbf{W} = [W(n) \ \ldots \ W(n+N-1)]^T$ to the vector obtained from N consecutive values of $W(n)$.

1. Write, as a function of $R_{WW}(k)$, the expression of the covariance matrix of $\mathbf{W}$.

2. Given an $(N \times N)$ matrix $\mathbf{M}$, let $\mathbf{X} = \mathbf{MW}$. Determine the expression of the covariance matrix of the vector $\mathbf{X}$ (definition 7.13). What is the probability distribution for $\mathbf{X}$ when $W(n)$ is a Gaussian process?

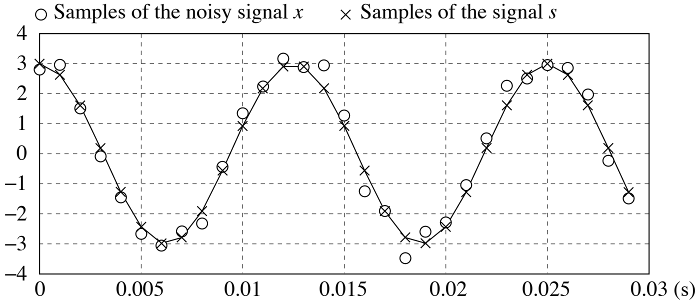


Figure 8.4 – *Sine with a frequency of* 80 *Hz, corrupted by white noise with a signal-to-noise ratio SNR* $= 15$ *dB*

3. Use this result to find a method for obtaining a sequence of N values with a given covariance, when $W(n)$ is white, zero-mean and with a variance equal to 1.

4. Use this result to find a method for obtaining a sequence N values of a white process with a variance equal to 1, when $X(n)$ has $R_{XX}(k)$ as its autocovariance function. This is called *whitening* the process $X(n)$.

HINT:

1. Because $W(n)$ is zero-mean, the covariance matrix is given by:

$$\begin{aligned}\mathbf{R}_W &= \mathbb{E}\{\mathbf{W}\mathbf{W}^H\} = [\mathbb{E}\{W(n+\ell)W^*(n+k)\}] \\ &= [R_{WW}(n+\ell-n-k)] = [R(\ell-k)]\end{aligned}$$

where $\ell, k \in \{0, \dots, N-1\}$, which leads us to:

$$\mathbf{R}_W = \begin{bmatrix} R_{WW}(0) & \cdots & R_{WW}(-N+1) \\ \vdots & \ddots & \vdots \\ R_{WW}(N-1) & \cdots & R_{WW}(0) \end{bmatrix}$$

2. If we start off with $\mathbf{X} = \mathbf{M}\mathbf{W}$, the covariance matrix has the expression:

$$\mathbf{R}_X = \mathbb{E}\{\mathbf{X}\mathbf{X}^H\} = \mathbb{E}\{\mathbf{M}\mathbf{W}\mathbf{W}^H\mathbf{M}^H\} = \mathbf{M}\mathbf{R}_W\mathbf{M}^H$$

If $W(n)$ is Gaussian, the sample $\mathbf{X}$ is Gaussian itself, since Gaussian nature is unchanged by linear transformation (see theorem 7.4). Its mean is zero and its covariance matrix is $\mathbf{R}_X$, meaning that the probability density has the expression:

$$p_{\mathbf{X}}(\mathbf{x}) = \frac{1}{(2\pi)^{N/2}\sqrt{\det(\mathbf{R}_X)}} \exp\left(-\frac{1}{2}\mathbf{x}^H\mathbf{R}_X^{-1}\mathbf{x}\right)$$

where $\mathbf{x} = [x_n, \cdots, x_{n+N-1}]^T$.

3. If the process $W(n)$ is white, zero-mean and has a variance of 1, its covariance matrix $\mathbf{R}_W = \mathbf{I}_N$, where $\mathbf{I}_N$ refers to the $N \times N$ identity matrix. Given the matrix $\mathbf{R}_X$, how should $\mathbf{M}$ be chosen for the sample $\mathbf{X} = \mathbf{MW}$, where $\mathbf{W}$ is white, to have the covariance $\mathbf{R}_X$? All we need to do is choose $\mathbf{M}$ such that $\mathbf{MM}^H = \mathbf{R}_X$. $\mathbf{M}$ is called *a square root* of $\mathbf{R}_X$. Just like in the case of scalars, a positive matrix has several square roots[4]. With MATLAB®, the `sqrtm(R)` function, with $\mathbf{R}$ positive, calculates the principal square root of $\mathbf{R}$.

4. Bear in mind that $\mathbf{R}_X = \mathbf{M}\mathbf{R}_W\mathbf{M}^H$. If we want $X(n)$ to be white with a variance of 1, $\mathbf{R}_X$ has to be equal to the identity matrix. This can be achieved by choosing $\mathbf{M}$ as the inverse of the square root of $\mathbf{R}_W$, which is obtained in MATLAB® using the command `inv(sqrtm(Rw))`.

   If $X(n)$ is Gaussian, then the sample $\mathbf{W}$ is Gaussian and white. If this is the case, we know (see theorem 7.3) that non-correlation implies independence. Therefore, the obtained sequence is comprised of independent variables.

■

This provides us with the following important result:

Colored noise can be changed into white noise by multiplying the sample by the inverse of the square root of the "colored" process's covariance matrix. This is called whitening the signal. *Furthermore, if the original samples are Gaussian, the processed samples are Gaussian and independent.*

8.2.3 Sampling a WSS process

Consider a *real, zero-mean*, WSS random process $X(t)$, $t \in \mathbb{R}$, with the PSD:

$$S(F) = \int_{-\infty}^{+\infty} R(\tau) e^{-2j\pi F\tau} d\tau$$

where $R(\tau) = \mathbb{E}\{X(t+\tau)X(t)\}$ represents its autocovariance function. Here, F is a frequency expressed in Hz, and (t, τ) are times expressed in seconds. We assume that $X(t)$ is *B-band limited*, meaning that $S(F) = 0$ for $|F| > B$.

[4] With scalars, if $r \geq 0$, the equation $mm^* = r$ has the solution $\sqrt{m}e^{j\phi}$ where ϕ is an arbitrary real number. $\sqrt{m}$ is called the positive square root, and the number $u = e^{j\phi}$ is such that $uu^* = 1$. Likewise, the matrix equation $\mathbf{MM}^H = \mathbf{R}$, where $\mathbf{R} \geq 0$, has an infinite number of solutions of the type $\sqrt{\mathbf{R}}\mathbf{U}$ where $\mathbf{U}$ is any unitary matrix that obeys $\mathbf{UU}^H = \mathbf{I}$.

The signal $X(t)$ is sampled at a frequency of $F_s = 1/T_s$. Its samples are denoted by $X_s(n) = X(nT_s)$. It can be shown [100] that if $F_s \geq 2B$, the process can be reconstructed, as a limit in quadratic mean, from its samples according to the reconstruction formula 2.24 proved in the deterministic case, the expression of which is recalled below:

$$X(t) = \sum_{n=-\infty}^{\infty} X_s(n) h_B(t - nT_s) \text{ where } h_B(t) = \frac{\sin(2\pi Bt)}{\pi F_s t} \tag{8.22}$$

In the case where $F_s < 2B$, perfect reconstruction is impossible because of aliasing.

As it was the case with deterministic signals, when a continuous-time WSS random process is sampled, the sampling operation at a frequency F_s must be preceded by anti-aliasing filtering with a gain of 1 in the $(-F_s/2, F_s/2)$ band to avoid aliasing.

We are now going to determine the relation between the PSD $S(F)$ of the continuous-time random process $X(t)$ and the PSD $S_s(f)$ of the random process sampled at $X_s(n) = X(n/F_s)$. Because of definition 8.13:

$$S_s(f) = \sum_k \mathbb{E}\{X_s(n+k)X_s(n)\}\, e^{-2j\pi kf}$$

If we use the fact that $X_s(n) = X(n/F_s)$, we get:

$$\mathbb{E}\{X_s(n+k)X_s(n)\} = \mathbb{E}\{X((n+k)/F_s)X(n/F_s)\} = R(k/F_s)$$

Replacing in $S_s(f)$ leads us to:

$$S_s(f) = \sum_k R(k/F_s) e^{-2j\pi kFT_s}$$

where we have assumed $F = fF_s$ and $F_sT_s = 1$. If we apply the identity given by the Poisson formula 2.4 to the second member, we get:

$$S_s(f) = F_s \sum_n S((f-n)F_s) \tag{8.23}$$

Finally, we find the PSD's expression for the process $X(t)$:

$$S(F) = \frac{1}{F_s} S_s(F/F_s) \mathbb{1}(F \in (-B, B)) \tag{8.24}$$

To sum up, it should be remembered that the PSD of $X(t)$ is obtained from the PSD of $X_s(n)$ by:

- multiplying the amplitude by $1/F_s$;
- multiplying the frequency axis by F_s;

- and by limiting the frequency band to the interval $(-B, B)$.

Conversely, the PSD of $X_s(n)$ is obtained from the PSD of $X(t)$ by:

- multiplying the amplitude by F_s;
- dividing the frequency axis by F_s;
- and by periodizing with the period 1.

To illustrate this, we are going to apply these formulae to the problem of the signal-to-quantization noise ratio when the signal is oversampled at frequency higher than the Nyquist frequency.

Quantization noise and oversampling

When oversampling a band-limited signal without quantizing it, we know, from the sampling theorem, that it is useless to oversample (compared with the Nyquist frequency). This changes completely when the sampling is followed by a quantization operation, because the quantization operation can be interpreted as the addition of noise. Let us see the consequences of oversampling in terms of signal-to-noise ratio.

Consider the B band, zero-mean, WSS, real random process $X(t)$. This signal is sampled at the frequency $F_s \geq 2B$, the sequence of obtained signals is denoted by $\{X_s(n)\}$. These samples are then uniformly quantized, with a quantization step q (see paragraph 7.5). Let $X_s^Q(n)$ be the samples that are quantized. We are going to reconstruct a signal denoted by $X^Q(t)$, from the quantized samples $X_s^Q(n)$, using expression 8.22, then try to evaluate the power of the "error" between the original signal $X(t)$ and the signal $X^Q(t)$ obtained from the quantized samples.

Starting off with 8.22, we can write successively:

$$X^Q(t) \quad = \quad \sum_n X_s^Q(n) h_B(t - nT_s)$$

where $h_B(t)$ is given by 2.24. Let us define $\varepsilon(n)$ with $X_s^Q(n) = X_s(n) + \varepsilon(n)$. We get:

$$X^Q(t) = \sum_n X_s(n) h_B(t - nT_s) + \sum_n \varepsilon(n) h_B(t - nT_s)$$

By hypothesis, $F_s \geq 2B$. Hence the first term is exactly equal to the signal $X(t)$ and therefore:

$$X^Q(t) = X(t) + \underbrace{\sum_n \varepsilon(n) h_B(t - nT_s)}_{=B^Q(t)}$$

The signal $B^Q(t)$ represents the error between the original signal and the reconstructed signal: it is called the *quantization noise.* Notice that its expression is obtained, from the process $\varepsilon(n)$, precisely by using the reconstruction formula 8.22. Therefore, according to expression 8.24, we can determine the PSD of $B^Q(t)$ from the PSD of $\varepsilon(n)$, and from there, determine its power.

By referring to the hypotheses on $\varepsilon(n)$ made in paragraph 7.5, we know that $\varepsilon(n)$ is a zero-mean random process with a variance of $q^2/12$ and such that $\mathbb{E}\{\varepsilon(n)\varepsilon(k)\} = 0$ for $n \neq k$. Hence the PSD of $\varepsilon(n)$ is given for any f by:

$$S_\varepsilon(f) = \frac{q^2}{12}$$

Using formula 8.24, the PSD of $B^Q(t)$ is then given by:

$$S_B(F) = \frac{q^2}{12}\frac{1}{F_s}\mathbb{1}(F \in (-B, B)) \tag{8.25}$$

The quantization noise's power is obtained by integrating $S_B(F)$:

$$P_B = \int_{-\infty}^{+\infty} S_B(F)dF = \frac{q^2}{12}\frac{2B}{F_s} \tag{8.26}$$

Using formula 7.34, we end up, in the case of uniform quantization with oversampling, with the following expression of the signal-to-noise ratio:

$$\text{SNR} = 6N + 10\log_{10}(3/F_c^2) + 10\log_{10}(F_s/2B) \tag{8.27}$$

where N refers to the number of bits of the quantizer and F_c to the clipping factor. A 3 dB gain occurs every time the sampling frequency is doubled. This result calls for a few comments:

1. According to the sampling theorem, oversampling is useless without the quantization operation. All the information useful to reconstructing the signal without errors is contained in the samples taken at $F_s = 2B$.

2. Formula 8.27 was obtained by assuming that the quantization noise is white (the sequence $\varepsilon(n)$ is uncorrelated). If this happens to be false, the PSD $S_B(F)$, given by expression 8.25, has a different shape (sharper peaks). This means that the quantization noise's power is no longer given by expression 8.26 and the gain can then be much less than 3 dB. This is the case when the oversampling factor becomes too high, because the non-correlation error hypothesis is not quite established anymore. Hence there cannot be an infinite iteration of the 3 dB gain by doubling the sampling frequency.

3. There is no point in interpolating (interpolating is not oversampling) the *already quantized* discrete-time sequence in the hope of obtaining samples

that would have been produced when oversampling a continuous-time signal. The errors introduced by the quantization process are permanently added, and the reconstructed samples are noised in the same way.

8.3 Estimating the covariance

The concept of ergodicity

In practice, the covariance functions are not known, and we are faced with the problem of estimating them. As we have already said, a random process can be seen as great number of trajectories corresponding to a great number of realizations of the identically repeated experiment. However, in many practical cases, we have at our disposal *only* one process trajectory. It then becomes clear that the stationary process category, for which the moments can be estimated by calculating a "temporal mean" on only one trajectory, will have an important practical role.

Ergodicity is related to this concept. However, we will not give its general definition here. We will only say that a WSS random process $X(n)$ with the mean $m = \mathbb{E}\{X(n)\}$ and the autocovariance function $R(k) = \mathbb{E}\{X_c(n+k)X_c^*(n)\}$, is *ergodic* if its mean and its autocovariance function can be obtained as the convergence in probability, when N tends to infinity, of a temporal mean calculated for only one trajectory. This can be expressed, when N tends to infinity:

$$\widehat{m}_N = \frac{1}{N}\sum_{n=0}^{N-1} X(n) \longrightarrow m \tag{8.28}$$

For the covariance, this leads to:

$$\widehat{R}_N(k) = \frac{1}{N}\sum_{n=0}^{N-k-1} (X(n+k) - \widehat{m}_N)(X^*(n) - \widehat{m}_N^*)) \longrightarrow R(k) \tag{8.29}$$

This convergence actually is not all that surprising. We know, for example, from the law of large numbers [28], that, for a sequence of independent random variables, with the same mean m and the same finite variance, which is a particular case of a WSS process, the empirical mean:

$$\frac{1}{N}\sum_{n=0}^{N-1} X(n)$$

converges in probability to m. The question is "does this result apply to a larger class of random processes than just the sequences of independent random variables, such as for example the WSS random processes?" The answer is yes

[82], but it is not fundamentally useful for what we are going to do to go into it any further. We will simply assume that, for the WSS processes we will be considering, the conditions are in fact met. We can then use expressions 8.28 and 8.29 to estimate the mean and the autocovariance function of a WSS process from the observation of N samples.

Notice that if the mean of $X(n)$ is m, then we can write that $X(n) = m + B(n)$, where $B(n)$ is a zero-mean process. If we start off with this, formula 8.29 for estimating covariance consists of estimating m first, then of subtracting this estimation to $X(n)$, and finally of estimating the covariance of $B(n)$. Generally speaking, we have to consider $X(n) = s(n;\theta) + B(n)$ where $B(n)$ is a zero-mean random process and $s(n;\theta)$ represents a deterministic signal that depends on a parameter θ we have to determine. In this context, $s(n;\theta)$ is sometimes referred to as the *trend* term. This trend can be either affine, polynomial or periodic. For the latter, the trend is said to be *seasonal*. In conclusion, estimating the covariance is achieved on the process $B(n)$, which is obtained in the following way:

– if a non-zero mean is observed, center the process by calculating $B(n) = X(n) - \frac{1}{N}\sum_{k=0}^{N-1} X(k)$;

– if an affine trend is observed, of the kind $s(n;\theta) = a_1 + a_2 n$ (here $\theta = (a_1, a_2)$), use the program written in exercise 8.1, that allows you to estimate the pair (a_1, a_2) and then to obtain the residue $B(n)$;

– if, finally, a seasonal trend is observed, of the kind $s(n;\theta) = a + b\cos(2\pi f_0 n - \phi)$ (here $\theta = (a, b, \phi)$ and f_0 is known), use the program written in example 8.6.

Suppressing a mean

With MATLAB®, the estimated mean $\frac{1}{N}\sum_{k=1}^{N} X(k)$ is obtained with the `mean` command. To obtain the zero-mean process, all you need to do is type `xc=x-mean(x)`. You can also type `moyx=sum(x)/N`, where `x` is assumed to be a length N column vector, then `xc=x-moyx`.

Exercise 8.1 (Suppressing an affine trend)

Consider a discrete-time random process $X(n) = a_1 + a_2 n + B(n)$ where the noise $B(n)$ is a centered WSS random process. What happens is that in the absence of noise, we get a line of equation $Y(n) = a_1 + a_2 n$, whereas in the presence of noise, we get a scatter plot, more or less spread out around this line. The problem will be to find the line that best fits the scattered points, the meaning of which will soon become clear.

To do this, we start with the observation of $X(n)$ over a time interval $\{0, \ldots, N-1\}$, and we assume that $B(n)$ is a white, Gaussian, random process with an unknown variance σ_b^2. a_1 and a_2 are the two unknown parameters we are going to determine.

1. Give the expression of the probability density $p_{\mathbf{X}}(x_0, \ldots, x_{N-1}; a_1, a_2, \sigma_b^2)$ of the random vector $\{X(0), \ldots, X(N-1))\}$ as a function of the parameters a_1, a_2 and σ_b^2.

2. Gauss had the idea of choosing the values of a_1, a_2 and σ_b^2 such that $p_{\mathbf{X}}(x_0, \ldots, x_{N-1}; a_1, a_2, \sigma_b^2)$ would be maximum. a_1, a_2 and σ_b^2 are called the *maximum likelihood* estimators. We choose the following notations:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}, \ \mathbf{X} = \begin{bmatrix} X(0) \\ \vdots \\ X(N-1) \end{bmatrix} \text{ and } \mathbf{W} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ \vdots & \vdots \\ 1 & N-1 \end{bmatrix}$$

   Determine, as a function of $\mathbf{X}$ and $\mathbf{W}$, the expression of $\mathbf{a}$ that maximizes the likelihood.

3. Write a function that eliminates the affine trend and only keeps the stationary part zero-mean. Test this function.

COMMENTS:

- What exercise 8.1 teaches us is that in the case of a Gaussian hypothesis, the maximum likelihood estimator coincides with the least square estimator.
- The method explained here can easily be applied to any polynomial trend of the kind $X(n) = a_0 + a_1 n + \cdots + a_N n^N + B(n)$.

Example 8.6 (Suppressing a seasonal trend)
In many fields, such as meteorology, economics and biology, certain behaviors show a periodicity related to natural phenomena that are periodic themselves: the rotation of the Earth around the Sun, the rotation of Earth on its axis, etc. These behaviors are said to show a *seasonal trend.* A model can then be used to describe them as a sum of deterministic components representing this trend and of a zero-mean random process $B(n)$, representing the variability of the phenomenon, which can be written:

$$X(n) = a + b\cos(2\pi f_0 n - \phi) + B(n)$$

In this context, the frequency f_0 is assumed to be known. Studying the process $B(n)$ requires a preprocessing to eliminate the seasonal trend. In order to do this, we estimate a, b, and ϕ, then subtract $a + b\cos(2\pi f_0 n - \phi)$ to $X(n)$ to obtain an estimation of the residual process $B(n)$. A criterion often used for estimation is the one called the *least squares* criterion (see the following comment in exercise 8.1). Beginning with the observation of $X(n)$ for $n \in \{0, \ldots, N-1\}$, we are trying to find the values of a, b and ϕ that minimize the RMS deviation:

$$J(a, b, \phi) = \sum_{n=0}^{N-1} (X(n) - (a + b\cos(2\pi f_0 n - \phi)))^2$$

between $X(n)$ and the expected seasonal evolution.

1. Determine the expressions of a, b and ϕ that minimize $J(a, b, \phi)$.
2. Write a function that eliminates the seasonal trend, leaving only the stationary part zero-mean.

HINT:

1. Let $\beta_1 = b\cos\phi$ and $\beta_2 = b\sin\phi$. We have:

$$\begin{aligned} J(a, \beta_1, \beta_2) &= \sum_{n=0}^{N-1} (X(n) - (a + b\cos(2\pi f_0 n - \phi)))^2 \\ &= \sum_{n=0}^{N-1} (X(n) - a - \beta_1 \cos(2\pi f_0 n) - \beta_2 \sin(2\pi f_0 n))^2 \end{aligned}$$

If we successively set to zero the derivatives with respect to a, β_1 and β_2, we get:

- with respect to a:

$$\sum_{n=0}^{N-1} (X(n) - a - \beta_1 \cos(2\pi f_0 n) - \beta_2 \sin(2\pi f_0 n)) = 0$$

- with respect to β_1:

$$\begin{aligned} &\sum_{n=0}^{N-1} X(n)\cos(2\pi f_0 n) - a\sum_{n=0}^{N-1} \cos(2\pi f_0 n) \\ &\qquad -\beta_1 \sum_{n=0}^{N-1} \cos^2(2\pi f_0 n) - \beta_2 \sum_{n=0}^{N-1} \sin(2\pi f_0 n)\cos(2\pi f_0 n) = 0 \end{aligned}$$

- with respect to β_2:

$$\begin{aligned} &\sum_{n=0}^{N-1} X(n)\sin(2\pi f_0 n) - a\sum_{n=0}^{N-1} \sin(2\pi f_0 n) \\ &\qquad -\beta_1 \sum_{n=0}^{N-1} \cos(2\pi f_0 n)\sin(2\pi f_0 n) - \beta_2 \sum_{n=0}^{N-1} \sin^2(2\pi f_0 n) = 0 \end{aligned}$$

Let:

$$\begin{aligned}
\mathbf{U} &= [1 \quad \dots \quad 1]^T \\
\mathbf{X} &= [X(0) \quad \dots \quad X(N-1)]^T \\
\mathbf{C} &= [\cos(2\pi f_0 \times 0) \quad \dots \quad \cos(2\pi f_0 \times (N-1))]^T \\
\mathbf{S} &= [\sin(2\pi f_0 \times 0) \quad \dots \quad \sin(2\pi f_0 \times (N-1))]^T
\end{aligned}$$

With these notations, we can group the three previous derivatives together to write a single matrix equation:

$$\begin{bmatrix} \mathbf{U}^T \\ \mathbf{C}^T \\ \mathbf{S}^T \end{bmatrix} [\mathbf{U} \quad \mathbf{C} \quad \mathbf{S}] \begin{bmatrix} a \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} \mathbf{U}^T \\ \mathbf{C}^T \\ \mathbf{S}^T \end{bmatrix} \mathbf{X}$$

which can also be written:

$$\begin{bmatrix} N & \mathbf{U}^T\mathbf{C} & \mathbf{U}^T\mathbf{S} \\ \mathbf{C}^T\mathbf{U} & \mathbf{C}^T\mathbf{C} & \mathbf{C}^T\mathbf{S} \\ \mathbf{S}^T\mathbf{U} & \mathbf{S}^T\mathbf{C} & \mathbf{S}^T\mathbf{S} \end{bmatrix} \begin{bmatrix} a \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} \mathbf{U}^T \\ \mathbf{C}^T \\ \mathbf{S}^T \end{bmatrix} \mathbf{X} \tag{8.30}$$

If $\mathbf{M}$ refers to the 3×3 matrix found in the left-hand side of equation 8.30, and if we assume $\mathbf{M}$ to be invertible:

$$\begin{bmatrix} a \\ \beta_1 \\ \beta_2 \end{bmatrix} = \mathbf{M}^{-1} \begin{bmatrix} \mathbf{U}^T \\ \mathbf{C}^T \\ \mathbf{S}^T \end{bmatrix} \mathbf{X} \tag{8.31}$$

In the case where $f_0 N \gg 1$, it can easily be checked that we successively have $\mathbf{U}^T\mathbf{C} \approx 0$, $\mathbf{U}^T\mathbf{S} \approx 0$, $\mathbf{S}^T\mathbf{C} \approx 0$, $\mathbf{C}^T\mathbf{C} \approx N/2$ and $\mathbf{S}^T\mathbf{S} \approx N/2$. This means that $\mathbf{M} \approx \text{diag}(N, N/2, N/2)$ for which we infer the following approximate expressions:

$$\begin{aligned}
a &\approx \frac{1}{N} \sum_{n=0}^{N-1} X(n) \\
\beta_1 &\approx \frac{2}{N} \sum_{n=0}^{N-1} X(n) \cos(2\pi f_0 n) \\
\text{and } \beta_2 &\approx \frac{2}{N} \sum_{n=0}^{N-1} X(n) \sin(2\pi f_0 n)
\end{aligned}$$

We will see in Chapter 11, on the least squares method, a generalization of this result to the sum of several periodic components.

2. Save the function trendseason.m:

```
function dx=trendseason(x,f0)
%%================================%
%% Suppressing a seasonal trend    %
%% SYNOPSIS: dx=TRENDSEASON(x,f0)  %
%%     x  = Input sequence         %
%%     f0 = Seasonal frequency     %
%%     dx = Residue                %
%%================================%
x=x(:); N=length(x);
U=ones(N,1);
C=cos(2*pi*f0*(0:N-1)'); S=sin(2*pi*f0*(0:N-1)');
M=[ N U'*C U'*S ; C'*U C'*C C'*S; S'*U S'*C S'*S];
theta=inv(M)*[U';C';S']*x;
dx=x-[U C S]*theta;
return
```

Test the tendseason function by executing the following program:

```
%===== TESTTRENDSEASON.M
N=100; B=randn(N,1); a=4; f0=0.01; phi=pi/6;
tseason=3*cos(2*pi*f0*(0:N-1)'-phi);
X=a+tseason+B; Res=trendseason(X,f0);
subplot(311); plot(B); grid; set(gca,'ylim',[-4 4])
subplot(312); plot(X); grid;
subplot(313); plot(Res); grid; set(gca,'ylim',[-4 4])
```

Because $f_0 N = 1$, the terms that do not belong to the diagonal of matrix $\mathbf{M}$ (see expression 8.31) are not negligible. You can check that the approximated formulas provide results with noticeable differences. ■

Estimating covariance

From now on, we will assume, except if specified otherwise, that the observation sequence has been previously processed in order to *remove the mean and the possible tendencies.* This means, according to expression 8.29, that the estimation of the autocovariance function from $\{X(0), \ldots, X(N-1)\}$ is given, for $k \in \{0, \ldots, K-1\}$, by:

$$\widehat{R}_{XX}(k) = \frac{1}{N}\sum_{n=0}^{N-k-1} X(n+k)X^*(n) = \frac{1}{N}\sum_{m=k}^{N-1} X(m)X^*(m-k) \qquad (8.32)$$

Likewise, an estimation of the covariance function between two random processes $X(n)$ and $Y(n)$, both assumed to be WSS and zero-mean, is given for $k \in \{0, \ldots, K-1\}$ by:

$$\widehat{R}_{YX}(k) = \frac{1}{N}\sum_{n=0}^{N-k-1} Y(n+k)X^*(n) = \frac{1}{N}\sum_{m=k}^{N-1} Y(m)X^*(m-k) \qquad (8.33)$$

From a theoretical point of view, it can be shown, as we said at the beginning of this paragraph, that for a very large category of WSS processes, the estimators given by expressions 8.32 and 8.33 converge, when N tends to infinity, to the true covariance [14].

In practice, k must however remain much smaller than the number N of observations. A practical rule is to choose k less than $N/10$.

Positivity of the estimated covariance matrix

Consider the first K values $\widehat{R}_{XX}(0), \ldots, \widehat{R}_{XX}(K-1)$ obtained with expression 8.32. To construct an estimation of the covariance matrix of a WSS process, you only need the hermitian Toeplitz matrix, for which the elements of the first column are precisely $\widehat{R}_{XX}(0), \ldots, \widehat{R}_{XX}(K-1)$. The matrix can be written:

$$\widehat{\mathbf{R}} = \begin{bmatrix} \widehat{R}_{XX}(0) & \widehat{R}_{XX}(-1) & \ldots & \widehat{R}_{XX}(-K+1) \\ \widehat{R}_{XX}(1) & \widehat{R}_{XX}(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \widehat{R}_{XX}(-1) \\ \widehat{R}_{XX}(K-1) & \ldots & \widehat{R}_{XX}(1) & \widehat{R}_{XX}(0) \end{bmatrix} \tag{8.34}$$

A simple calculation shows that:

$$\widehat{\mathbf{R}} = \frac{1}{N}\mathbf{D}^H\mathbf{D} \tag{8.35}$$

with $\mathbf{D}^H =$

$$\begin{bmatrix} X^*(0) & X^*(1) & \cdots & X^*(N-1) & 0 & \cdots & 0 \\ 0 & X^*(0) & X^*(1) & \cdots & X^*(N-1) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \ddots & 0 \\ 0 & \cdots & 0 & X^*(0) & X^*(1) & \cdots & X^*(N-1) \end{bmatrix} \tag{8.36}$$

The fact that $\widehat{\mathbf{R}}$ can be written as $\mathbf{D}^H\mathbf{D}/N$ guarantees that $\widehat{\mathbf{R}}$ is both *hermitian and positive*. This is because for any vector $\mathbf{a}$, we can write that:

$$\mathbf{a}^H\widehat{\mathbf{R}}\mathbf{a} = \frac{1}{N}\mathbf{a}^H\left(\mathbf{D}^H\mathbf{D}\right)\mathbf{a} = \frac{1}{N}(\mathbf{D}\mathbf{a})^H(\mathbf{D}\mathbf{a}) = \frac{1}{N}\mathbf{v}^H\mathbf{v}$$

where we let $\mathbf{v} = \mathbf{D}\mathbf{a}$. We can conclude by noticing that the scalar $\mathbf{v}^H\mathbf{v}$ is the sum of the square moduli of the components of $\mathbf{v}$, and is therefore positive. Obviously, in practice, you do not construct $\mathbf{D}$ to then calculate $\mathbf{D}^H\mathbf{D}$. You calculate $\widehat{R}_{XX}(k)$ for $k = 0, \ldots, K-1$ with formula 8.32, then you use the `toeplitz` function of MATLAB® to store the obtained values in a matrix of the type 8.34.

In the literature, this method for calculating $\widehat{\mathbf{R}}$ is called the *correlation method*. What we see is that, in a way, everything is as if we had padded the observed sequence on the left and on the right with $(K-1)$ zeros. Its major drawback is therefore to add false data, zeros to be precise, on both sides of the observed data. This is why when the length N of the sample is small, it is usually discarded, to the benefit of the *covariance method* which consists of choosing as the covariance matrix:

$$\widehat{\mathbf{R}} = \frac{1}{N-K}\mathbf{D}^H\mathbf{D}$$

with:

$$\mathbf{D}^H = \begin{bmatrix} X^*(K-1) & X^*(K) & \cdots & X^*(N-1) \\ X^*(K-2) & X^*(K-1) & \cdots & X^*(N-2) \\ \vdots & \vdots & & \vdots \\ X^*(0) & X^*(1) & \cdots & X^*(N-K) \end{bmatrix} \tag{8.37}$$

The resulting covariance matrix remains, of course, positive, but it loses its Toeplitz structure necessary to certain fast inversion algorithms.

There are two other methods for constructing $\mathbf{D}$, padding with zeros either on the left or on the right. For example:

$$\mathbf{D}^H = \begin{bmatrix} X^*(K-1) & \cdots & X^*(N-1) & 0 & \ldots & 0 \\ X^*(K-2) & X^*(K-1) & \cdots & X^*(N-1) & 0 & \vdots \\ \vdots & \ldots & \ldots & \ddots & \ddots & 0 \\ X^*(0) & X^*(1) & \cdots & & & X^*(N-1) \end{bmatrix}$$

We give in paragraph 9.2.1 a comparison of these methods. If $N \gg K$ it is obvious that they will give basically the same results.

COMMENT: in the correlation calculation, each of the terms can be interpreted as a convolution, hence the idea to use the DFT for calculating the sequence of the covariance. We know that with a convolution in the time domain corresponds a product in the frequency domain, and the estimated autocovariance function can be seen precisely as the convolution of $x(n)$ with $x^*(-n)$, with which $X(f)X^*(f) = |X(f)|^2$ corresponds by Fourier transform. However, bear in mind that in the DFT context, we know that the associated convolution is *circular*.

8.4 Filtering formulae for WSS random processes

Filtering formula for the PSD

Let $X(t)$ be a WSS random process, with the autocovariance function $R_{XX}(\tau)$ and the PSD $S_{XX}(f)$ fed into the input of a linear filter with the impulse response $h(t)$ and the complex gain $H(f)$.

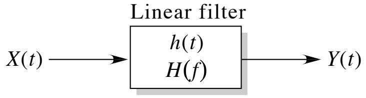


Figure 8.5 – *Linear filter*

We assume that $h(t)$ is summable (BIBO stable filter). In the continuous-time case, this can be expressed:

$$\int_{\mathbb{R}} |h(t)|dt < +\infty$$

and in the discrete-time case[5]:

$$\sum_{t\in\mathbb{Z}} |h(t)| < +\infty$$

It can be proven [14] that the output random process $Y(t)$ is WSS itself. Its mean is given by:

$$m_Y = m_X H(0) \tag{8.38}$$

Therefore, it is zero-mean if the input signal is zero-mean. Its PSD is given by the following expression:

$$S_{YY}(f) = |H(f)|^2 S_{XX}(f) \tag{8.39}$$

Notice that if we decide to use the distribution formalism, formula 8.39 can still be applied when the process is harmonic. Consider for example as the input signal of the filter the real harmonic process $X(t) = \sum_{k=1}^{P} A_k \cos(2\pi f_k t + \Phi_k)$, the PSD of which is given according to 8.19 by:

$$S_{XX}(f) = \frac{1}{4}\sum_{k=1}^{P} \sigma_k^2 \delta(f + f_k) + \frac{1}{4}\sum_{k=1}^{P} \sigma_k^2 \delta(f - f_k)$$

A direct calculation of the output signal $Y(t)$ leads to:

$$Y(t) = \sum_{k=1}^{P} A_k H(-f_k) e^{-2j\pi f_k t+\Phi_k} + \sum_{k=1}^{P} A_k H(f_k) e^{2j\pi f_k t+\Phi_k}$$

If we use 8.19, we get, for the PSD of $Y(t)$:

$$\begin{aligned} S_{YY}(f) &= \frac{1}{4}\sum_{k=1}^{P} \sigma_k^2 |H(-f_k)|^2 \delta(f + f_k) \\ &+\frac{1}{4}\sum_{k=1}^{P} \sigma_k^2 |H(f_k)|^2 \delta(f - f_k) \end{aligned} \tag{8.40}$$

[5] In this paragraph, t and τ belong either to $\mathbb{R}$ or to $\mathbb{Z}$ depending on whether the considered process is continuous-time or discrete-time.

which can be identified with $S_{YY}(f) = |H(f)|^2 S_{XX}(f)$ if we use the identity $H(f)\delta(f - f_0) = H(f_0)\delta(f - f_0)$, where $\delta(f)$ refers to the Dirac distribution. Starting off with 8.40, we end up with the following expression of the autocovariance function:

$$R_{YY}(\tau) = \frac{1}{4}\sum_{k=1}^{P} \sigma_k^2 |H(-f_k)|^2 e^{-2j\pi f_k \tau} + \frac{1}{4}\sum_{k=1}^{P} \sigma_k^2 |H(f_k)|^2 e^{2j\pi f_k \tau}$$

If the filter is real, $H(-f_k) = H^*(f_k)$ and:

$$R_{YY}(\tau) = \frac{1}{2}\sum_{k=1}^{P} \sigma_k^2 |H(f_k)|^2 \cos(2\pi f_k \tau)$$

In the case where the input process is a white noise, the PSD $S_{XX}(f)$ is constant. It may be useful to write the *Parseval formula* again:

$$\begin{cases} \int_{-\infty}^{+\infty} |h(t)|^2 dt = \int_{-\infty}^{+\infty} |H(f)|^2 df & \text{(continuous time)} \\ \sum_{t=-\infty}^{+\infty} |h(t)|^2 = \int_{-1/2}^{+1/2} |H(f)|^2 df & \text{(discrete time)} \end{cases} \tag{8.41}$$

which allows you to calculate the filter's output power by integrating the impulse response either in time or in frequency.

The output autocovariance function's expression is not as simple as the PSD's. If we restrict ourselves to the discrete-time case, we have:

$$R_{YY}(\tau) = \sum_{m=-\infty}^{+\infty} \left(\sum_{n=-\infty}^{+\infty} h(n)h^*(n-m) \right) R_{XX}(\tau - m) \tag{8.42}$$

Filtering formulae for the interspectrum

If the previous stationarity hypotheses are made, it can be proven that the processes $X(t)$ and $Y(t)$ have *stationary covariance*, meaning that $\mathbb{E}\{Y_c(t+\tau)X_c^*(t)\} = R_{YX}(\tau)$ (the index c refers to zero-mean processes) only depends on the time gap τ. Once again this formula has an simpler expression in frequency. We have:

$$S_{YX}(f) = H(f)S_{XX}(f) \tag{8.43}$$

where $S_{YX}(f)$, which is called the *interspectrum*, refers to the Fourier transform of $R_{YX}(\tau)$. Note that this function has none of the PSD's remarkable properties. In particular, there is no reason why it should be positive, or even real. If we restrict ourselves to the case of *discrete-time* random processes, we infer:

$$R_{YX}(\tau) = \mathbb{E}\{Y_c(t+\tau)X_c^*(t)\} = \sum_{m=-\infty}^{+\infty} h(m)R_{XX}(\tau - m) \tag{8.44}$$

Notice that if $X(t)$ is white with a variance of 1, that is $R_{XX}(\tau) = \delta(\tau)$, formula 8.44 can be simplified, and leads to:

$$R_{YX}(\tau) = h(\tau)$$

The impulse response coincides with the output/input covariance. This result can be used for estimating a filter's impulse response (see exercise 8.5).

Exercise 8.2 (Smoothing filtering of noise)
Consider the filter $h(n) = 1/8$ for $0 \leq n \leq 7$ and 0 otherwise. A white, zero-mean random process with a variance of 1 is fed into the input.

1. Determine the gain $|H(f)|^2$ of the filter.
2. Use this result to find the output process's spectrum and the output power.
3. Use this result to find the form of the output autocovariance function. Determine after which value k the output autocovariance function is null.
4. Write a program that simulates the filtering over 2,000 points of data, that evaluates the output autocovariance function using formula 8.32, and uses this to find the spectrum by an FFT calculation over 512 points.
5. Compare with the theoretical results.

Generating a random signal using white noise

We are often faced with the problem of simulating the trajectory of a WSS process with a given spectrum. The functions `randn` and `filter` make it possible to construct such a trajectory.

The `randn` function generates samples of Gaussian white noise with a variance of 1. Its spectrum is therefore constant, and equal to 1 in the $(-1/2, +1/2)$ band. The WSS process filtering formula 8.39 shows that we can obtain the trajectory of the process $Y(n)$ with a given spectrum, by properly filtering $W(n)$. We have:

$$S_{YY}(f) = |H(f)|^2 \times S_{WW}(f) = |H(f)|^2$$

Usually, the complex gain $H(f) = H_z(e^{2j\pi f})$ is that of a filter whose transfer function is a rational function that can be written $H_z(z) = B_z(z)/A_z(z)$. This means we have to use the `filter` function with the command `y=filter(b,a,w)`, where `a` and `b` refer to the denominator and numerator polynomials of the transfer function.

When used like this, the `filter` function starts with zero initial conditions, creating at the beginning of the trajectory a transient part that does not correspond exactly to the intended trajectory. One way of partly avoiding this is to spread out the first P values of the obtained signal, when the choice of P is directly related to the duration of the transient state of the filter's impulse response. For the numerator of the transfer function, this duration is simply the number of coefficients. For the denominator, we know that this value is related to the position of the poles with respect to the unit circle. A simple and practical rule consists of considering the modulus ρ_{max} of the most resonant pole, that is the one closest to the unit circle, and to choose P such that ρ_{max}^P is negligible compared with the root-mean-square of $Y(n)$. Remember that for a pole with the modulus ρ, the transient state decreases like ρ^n (see impulse response of a filter on page 128).

Example 8.7 (Generating a random signal)
Consider the process $x(n)$ obtained as the output of the filter with the transfer function $H_z(z) = 1/(1 + az^{-1})$, where a is a real number with its modulus less than 1, the white noise $W(n)$ with a variance of 1 being fed to its input.

1. Determine its PSD's expression.

2. Write a program that generates a trajectory for the process $X(n)$.

HINT:

1. Its PSD's expression is given by formula 8.39:

$$S_{XX}(f) = |H(f)|^2 = \frac{1}{(1 + ae^{2j\pi f})(1 + ae^{-2j\pi f})} = \frac{1}{1 + 2a\cos(2\pi f) + a^2}$$

2. Type:

```
%===== AR1.M
a=0.9; N=1000; W=randn(N,1);
X=filter(1,[1 a],W); plot(X); grid
```

Note that $0.9^{100} = 2.7 \times 10^{-5}$ is negligible. Hence we can consider that after the hundredth sample the obtained signal almost represents the trajectory of a stationary random process. ■

Exercise 8.3 (Generating a band limited process)
Write a program that generates $T = 1{,}000$ samples of a real random process sampled at a frequency of 10,000 Hz, the PSD of which is constant in the ($-1{,}000$ Hz - 1,000 Hz) band and null beyond it. Its power is assumed to be 2 Watts.

Exercise 8.4 (Pre-emphasis and de-emphasis)
When transmitting a signal through a channel subjected to noise, the signal-to-noise ratio can be enhanced by adding to the emitter a filter $H_p(f)$, called

the *pre-emphasis* filter, and to the receiver the inverse filter $H_d(f) = 1/H_p(f)$, called the *de-emphasis* filter. The choice of $H_p(f)$ depends on the spectral properties of the signal and of the noise.

Consider the real, zero-mean, second order stationary discrete-time process $X(n)$, we will assume that its PSD $S_X(f)$ is known. This signal is corrupted by a zero-mean, WSS, additive noise $B(n)$ for which the PSD $S_B(f)$ is also assumed to be known (Figure 8.6). This type of situation is encountered in transmission channels of communication systems, but also in any processing that adds noise to the signal, such as the quantization operation

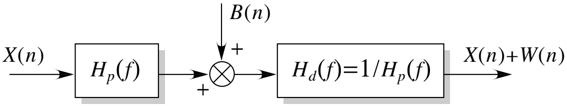


Figure 8.6 – *Pre-emphasis and de-emphasis system*

The output signal has the expression $X(n) + W(n)$ where $W(n)$ refers to the noise obtained by filtering the noise $B(n)$. The goal is to determine $H_p(f)$ (and hence $H_d(f) = 1/H_p(f)$ also) so as to minimize the power of $W(n)$. But a constraint has to be imposed, since we can make the power of $W(n)$ as small as we want it to be: we only have to multiply $H_p(f)$ by a very large factor A and to divide $H_d(f)$ by that same factor A, thus dividing the power of $W(n)$ by A^2 while leaving $X(n)$ untouched. This is why we are going to compare the SNRs obtained *with* and *without* the pre-emphasis/de-emphasis system. P_0 denotes the output power of the filter $H_p(f)$.

1. Determine as a function of $H_d(f)$ and $S_B(f)$ the expression of the power of the signal $W(n)$. Use this result to find the expression of the signal-to-noise ratio ρ_{PD} of the system represented in Figure 8.6.

2. Determine as a function of $H_p(f)$ and $S_X(f)$ the expression of P_0. Use this result to find the expression of the SNR $\rho = P_0/E(|B(n)|^2)$ for a process that does not use the suggested pre-emphasis/de-emphasis system.

3. Let $g = \rho_{\mathrm{PD}}/\rho$. The factor g can be interpreted as a gain: the higher it is, the better the suggested system. Determine the expression of g.

4. Using the Schwarz inequality, determine the expression of the filter $H_p(f)$ for which g is maximum.

Exercise 8.5 (Estimation of an FIR filter's impulse response)
As you may remember, the formula 8.44 giving the output/input covariance is:

$$R_{YX}(k) = \mathbb{E}\{Y_c(n+k)X_c^*(n)\} = h(k) \star R_{XX}(k) \tag{8.45}$$

In the case where $X(n)$ is a white process with the PSD σ^2, the expression can be simplified, leading to $R_{YX}(k) = \sigma^2 h(k)$. In this exercise, we are going to use this result to estimate the impulse response of an FIR filter.

1. Determine again the expression we found relating the output/input covariance function $R_{YX}(k)$ to the input autocovariance function $R_{XX}(k)$, for a linear filter with a finite impulse response of length L, assumed to be known. Show that the vector $\mathbf{h} = [h(0) \ \ldots \ h(L-1)]^T$ is the solution to a matrix expression of the type $\mathbf{Rh} = \mathbf{r}$ where $\mathbf{R}$ is a matrix constructed from $R_{XX}(k)$ and $\mathbf{r}$ is a vector constructed from $R_{YX}(k)$.

2. Write a program that performs an estimation of $\mathbf{h}$ based on the values of X and Y generated by the following program:

```
%===== GENEREPIMP.M
tps=(-16:1.2:15); h=sin(tps*(pi/5.8)) ./ tps*(pi/5.8);
num=[0.3 0.4 -0.2 0.1]; den=[1 -0.8 +0.5];
x=filter(num,den,randn(1,300)); y=filter(h,1,x);
```

   This method, called the *method of moments* should be compared with the least squares method (see Chapter 11).

8.5 MA, AR and ARMA time series

The search for models to describe random processes is at the core of signal processing, and the applications cover most of the applied fields. In this section, we will discuss models originating from the linear filtering of a white noise, and only discrete-time processes will be considered.

8.5.1 Q order MA (*Moving Average*) process

Definition 8.10 (MA-Q process) *An MA-Q process, MA for Moving Average, is the random process defined by:*

$$X(n) = W(n) + b_1 W(n-1) + \cdots + b_Q W(n-Q) \tag{8.46}$$

where $W(n)$ refers to a centered, second order stationary, white random process with a variance of σ^2 and $\{b_1, \ldots, b_Q\}$ is a sequence of Q coefficients.

The process constructed in this manner turns out to be the mean weighted by the sequence $\{1, b_1, \ldots, b_Q\}$ of the last $(Q+1)$ input values. Everything happens as if this weighting sequence was applied to the input signal, which is why the process is called *Moving Average*. The process $X(n)$ can also be seen as the output of a linear filter the impulse response of which is the sequence $\{1, b_1 \ldots, b_Q\}$. Therefore, this FIR filter has the following transfer function:

$$B(z) = 1 + b_1 z^{-1} + \cdots + b_Q z^{-Q} \tag{8.47}$$

It has no poles, hence it is stable.

The `filter` function can be used to obtain the trajectory for such a process. The following program generates 300 samples of an MA-2 process where $b_1 = 1.5$, $b_2 = -1.2$ and $\sigma^2 = 1$:

```
%===== TRAJMA.M
B=[1 1.5 -1.2]; w=randn(1,300);
x=filter(B,1,w); plot(x); grid
```

Relations between the model's parameters and the covariances for an MA-Q

We are going to determine the relations between the model's parameters and the covariance of the process $X(n)$, starting with the example of an (MA-2) process associated to the equation:

$$X(n) = W(n) + b_1 W(n-1) + b_2 W(n-2)$$

where $W(n)$ is a white, centered, WSS random process with the variance σ^2. First, we have $\mathbb{E}\{X(n)\} = 0$. Hence the process is centered. Let us determine the expression of $R(k) = \mathbb{E}\{X(n+k)X^*(n)\}$ as a function of b_1, b_2 and σ^2. Using linearity and the fact that $W(n)$ is white, we successively get:

$$\begin{cases} R(k) = \mathbb{E}\{X(n+k)X^*(n)\} = 0 \quad \text{for} \quad k \leq -3 \\ R(-2) = \mathbb{E}\{X(n-2)X^*(n)\} = \sigma^2 b_2^* \\ R(-1) = \mathbb{E}\{X(n-1)X^*(n)\} = \sigma^2(b_1^* + b_1 b_2^*) \\ R(0) = \mathbb{E}\{X(n)X^*(n)\} = \sigma^2(1 + |b_1|^2 + |b_2|^2) \\ R(1) = \mathbb{E}\{X(n+1)X^*(n)\} = \sigma^2(b_1 + b_2 b_1^*) \\ R(2) = \mathbb{E}\{X(n+2)X^*(n)\} = \sigma^2 b_2 \\ R(k) = 0 \quad \text{for} \quad k \geq 3 \end{cases} \tag{8.48}$$

This result can be generalized to any MA-Q process for which the autocovariance function has the expression (if we note that):

$$R(k) = \begin{cases} \sigma^2 \sum_{j=0}^{Q-|k|} b_{j+|k|} b_j^* & \text{for} \quad 0 \leq k \leq Q \\ \sigma^2 \sum_{j=0}^{Q-|k|} b_{j+|k|}^* b_j & \text{for} \quad -Q \leq k \leq 0 \\ 0 & \text{for} \quad |k| > Q \end{cases} \tag{8.49}$$

We check that $R(k) = R^*(-k)$. The sequence of covariances of an MA-Q has $2Q + 1$ non-zero terms. It is of the second degree with respect to the parameters b_j.

Obviously, equation 8.49 can be used for estimating the model's $(Q + 1)$ parameters, by substituting the autocovariance coefficients $R(k)$ with their estimates given by 8.32. Before we see an example, we are going to focus on the PSD of $X(n)$.

Spectrum of an MA-Q

According to equation 8.46, an MA-Q process can be seen as the output of a filter the input of which is a white noise with the PSD σ^2. Hence formula 8.39 can be applied and leads, for the PSD, to:

$$S(f) = \sigma^2 \left|1 + b_1 e^{-2j\pi f} + \cdots + b_Q e^{-2j\pi Q f}\right|^2 \tag{8.50}$$

COMMENT: shows 8.50 that knowing $S(f)$, which is equivalent, by definition, to knowing the covariance coefficients, only allows us to determine the modulus of $B(e^{2j\pi f})$. Because of theorem 4.6, we know that the roots of $B(z)$ can be inside as well as outside the unit circle without it changing the value of $|B(e^{2j\pi f})|$. Therefore, if we start with the covariance coefficients of the PSD, or in practice with their estimates, we have 2^Q solutions for the polynomial $B(z)$, all of them leading to the same spectrum $S(f)$. If this is all we know, there is no reason why one of them should be chosen rather than another. However, if we have reason to believe, in a particular problem, that $B(z)$ has all its roots inside the unit circle, that is if $B(z)$ is minimum phase (definition 4.8), then $B(z)$ can be identified. Unfortunately, in digital communications, this is never the case. Completely identifying $B(z)$ requires the use of what is called *higher order statistics*, or HOS (higher implicitly means higher than 2). This rules out the Gaussian case for good, since in that case, the HOS are statistical functions of the second order.

Example 8.8 (Minimum phase MA-1 process)
Consider a real MA-1 process defined by $X(n) = W(n) + b_1 W(n-1)$, where $W(n)$ is a white, centered, WSS random process with the variance σ^2.

1. Determine the sequence of the covariances.

2. The first order correlation coefficient is defined by:

   $$\rho = R(1)/R(0)$$

   Starting with the definition of the PSD, show that $|\rho| < 1/2$ (use the positive nature of the PSD).

3. Show that b_1 satisfies a second degree equation dependent on $R(0)$ and $R(1)$.

4. Assuming that $B(z)$ is minimum phase, show that only one of the two solutions is possible.

HINT:

1. We have $R(0) = \sigma^2(1 + b_1^2)$, $R(\pm 1) = \sigma^2 b_1$ and $R(k) = 0$ for $|k| \geq 2$.

2. By definition, the spectrum is:

$$S(f) = R(1)e^{2j\pi f} + R(0) + R(1)e^{-2j\pi f} = R(0)(1 + 2\rho\cos(2\pi f))$$

   The condition $S(f) \geq 0$ imposes that $|\rho| < 1/2$ (see example 8.3).

3. We get:

$$\rho b_1^2 - b_1 + \rho = 0$$

   This equation has two solutions: $b_1 = \frac{1+\sqrt{1-4\rho^2}}{2\rho}$ and $b_1' = \frac{1-\sqrt{1-4\rho^2}}{2\rho}$ the product of which is 1. Knowing ρ, or in the case of a sequence of N observations $X(1)$, ..., $X(N)$, knowing its estimate:

$$\hat{\rho} = \begin{cases} \min\left(\dfrac{\sum_{i=1}^{N-1} X_{i+1}X_i}{\sum_{i=1}^{N} X_i^2}, 1/2\right) & \text{if } \sum_{i=1}^{N-1} X_{i+1}X_i \geq 0 \\ \max\left(\dfrac{\sum_{i=1}^{N-1} X_{i+1}X_i}{\sum_{i=1}^{N} X_i^2}, -1/2\right) & \text{if } \sum_{i=1}^{N-1} X_{i+1}X_i < 0 \end{cases}$$

   these two roots are both as likely. This is another example of the problem expressed in the comment that follows expression 8.50.

4. We assume that $B(z)$ is minimum phase. In that case, $|b_1| \leq 1$, and therefore only one of the two solutions is possible.

■

Notice, finally, that the system of equations we wish to solve is not linear. This is why when estimating an MA, even a short one, it is usually preferable (see exercise 9.4) to approximate it with a long AR because, as we are going to see, the system is now linear.

8.5.2 P order AR (*Autoregressive*) Process

Consider the recursive equation:

$$X(n) + a_1 X(n-1) + \cdots + a_P X(n-P) = W(n) \qquad (8.51)$$

where $W(n)$ refers to a white, centered, WSS random process with the variance σ^2, and where $\{a_1, \ldots, a_P\}$ is a sequence of coefficients. If we let:

$$H_z(z) = \frac{1}{1 + a_1 z^{-1} + \cdots + a_P z^{-P}} \qquad (8.52)$$

then the signal $X(n)$ can be seen as the output of the all-pole filter with the transfer function $H_z(z)$ and the process $W(n)$ as its input (Figure 8.7).

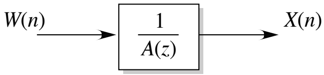


Figure 8.7 – *Generating an AR process*

It can be shown [14] that the recursive equation 8.51 has a single solution $X(n)$, second order stationary if and only if the denominator polynomial 8.52 is different from 0 for $|z| = 1$ (no poles on the unit circle). This results leads us to adopting the following definition:

Definition 8.11 (AR process) *A P order autoregressive process, or AR-P, is the only WSS process to the equation:*

$$X(n) + a_1 X(n-1) + \cdots + a_P X(n-P) = W(n) \tag{8.53}$$

where $W(n)$ refers to a centered, second order stationary, white random process with a variance of σ^2 and where the polynomial:

$$A(z) = 1 + a_1 z^{-1} + \cdots + a_P z^{-P} \neq 0 \text{ for } |z| = 1 \tag{8.54}$$

The expression of this solution is:

$$X(n) = \sum_{k=-\infty}^{+\infty} h_k W(n-k) \tag{8.55}$$

where h_k is the sequence of the Fourier series expansion coefficients of the function $H(f) = 1/A(e^{2j\pi f})$.

In the case where $A(z) \neq 0$ for $|z| \geq 1$, the poles of $H_z(z)$ are strictly inside the unit circle, $h_k = 0$ for $k < 0$ and $X(n)$ can be causally expressed as a function of $W(n)$:

$$X(n) = W(n) + h_1 W(n-1) + \cdots + h_k W(n-k) + \cdots \tag{8.56}$$

Notice that the stationary solution to equation 8.53 is the same as the stable solution we obtained in the case of deterministic signals (see theorem 4.3, page 113). Finally, remember that if $W(n)$ is Gaussian, then $X(n)$ itself is Gaussian since Gaussian nature is unchanged by linear transformations (see theorem 7.4).

Spectrum of an AR-P

Since an AR-P can be interpreted as the output of a filter fed with a white noise with the PSD σ^2, formula 8.39 can be used to determine the PSD's expression. This leads us to:

$$S(f) = \frac{\sigma^2}{|1 + a_1 e^{-2j\pi f} + \cdots + a_P e^{-2j\pi P f}|^2} \tag{8.57}$$

As an example, let us plot the trajectory of an AR-2 process associated with the polynomial $A(z) = 1 + a_1 z^{-1} + a_2 z^{-2}$, the two poles of which are conjugated, imposing that $a_1^2 - 4a_2 < 0$. Let ρ be the modulus of the two conjugated poles and $\pm\phi$ their respective phases. Starting off with ρ and ϕ, and by calculating the product and the sum of the roots, we get $a_2 = \rho^2$ and $a_1 = -2\rho\cos(\phi)$. The program `trajAR.m` displays a trajectory of this process.

```
%===== TRAJAR.M
sigma=2; phi=20*pi/180; rho=0.9;
a1=-2*rho*cos(phi); a2=rho*rho;
w=randn(1,300); x=filter(sigma,[1 a1 a2],w); plot(x); grid
```

Example 8.9 (Noised sine function versus AR-2)
Write a program:

- that displays a sequence of $N = 100$ samples taken at the frequency $F_s =$ 1,000 Hz, from a signal $Y(t) = s(t) + B(t)$, sum of a sine $s(t) = \sin(2\pi F_0 t)$ with a frequency of $F_0 = 100$ Hz and of a zero-mean, Gaussian, white noise $B(t)$ with variance of $\sigma_B^2 = 0.04$;

- that displays a sequence of $N = 100$ samples taken at the frequency $F_s = 1{,}000$ Hz, from an AR-2, solution of the equation $X(n) + a_1 X(n - 1) + a_2 X(n - 1) = W(n)$, where $W(n)$ is a white, centered, WSS process with the variance σ_W^2 and with a_1 and a_2 such that the filter has a resonance at $F_0 = 100$ Hz (see equation 4.26). Using 8.58, derive the value of σ_W^2 such that $X(n)$ and $Y(n)$ have the same power.

Compare the results.

HINT: type:

```
%===== SINUSVERSUSAR2.m
N=100; F0=100; FS=1000; tps=(0:N-1)/FS; sigmaB2=0.04;
y=sin(2*pi*F0*tps)+sqrt(sigmaB2)*randn(1,N);
%=====
a2=0.98; a1=-4*a2*cos(2*pi*F0/FS)/(1+a2); a=[1 a1 a2];
%===== X and Y have the same power
R0=1/2 +sigmaB2; R1=-R0*a1/(1+a2); R2=-a1*R1-a2*R0;
%=====
sigmaW=sqrt(R0+a1*R1+a2*R2); w=sigmaW*randn(1,N+1000);
x=filter(1,a,w); x=x(1001:length(x));
subplot(211); plot(tps,y); grid
subplot(212); plot(tps,x); grid
```

Figure 8.8 shows that the graph (a) presents irregularities but, however large the errors, periodogram analysis is applicable to such curve, and, given

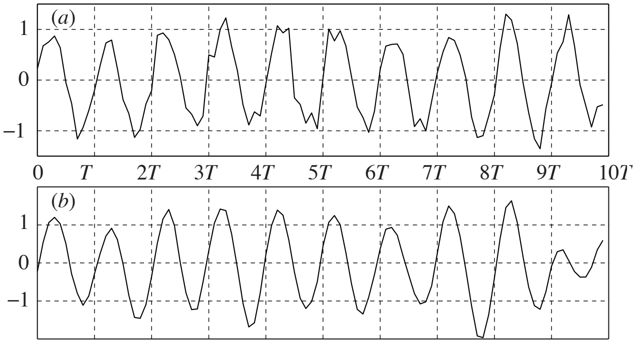


Figure 8.8 – *Noised sine function versus AR-2 process*

a sufficient number of periods, should yield a close approximation to the period. On the other hand, there are not abrupt variations in the graph (b), but the amplitude varies within wide limits, and the phase is continually shifting. Increasing the magnitude of $W(n)$ simply increases the amplitude: the graph remains smooth. ■

Relations between the model's parameters and the covariances for an AR-P

Property 8.3 *For a causal AR-P process, the relation between the model's parameters and the covariances $R(k)$, with $R(k) = R^*(-k)$, are given by:*

$$\begin{bmatrix} R(0) & R(-1) & \cdots & R(-P) \\ R(1) & R(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & R(-1) \\ R(P) & \cdots & R(1) & R(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ \vdots \\ a_P \end{bmatrix} = \begin{bmatrix} \sigma^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{8.58}$$

and for $k > P$ by:

$$R(k) = -\sum_{i=1}^{P} a_i R(k-i) \tag{8.59}$$

Equations 8.58 are called *normal equations* or *Yule-Walker equations.* We will see later on that they are directly related to the problem of linear prediction. More precisely, we will show that:

$$\hat{X}(n) = -\sum_{k=1}^{P} a_k X(n-k)$$

represents the best linear estimation in the least squares sense, of $X(n)$ based on its past and that the prediction error, defined by:

$$\varepsilon(n) = X(n) - \hat{X}(n) \tag{8.60}$$

is therefore equal to $W(n)$.

To establish relations 8.58 and 8.59, we start by multiplying the two members of the recurrence relation 8.53 by $X^*(n-k)$, then if we consider its mathematical expectation, we get:

$$\mathbb{E}\{(X(n) + \cdots + a_P x(n-P))X^*(n-k)\} = \mathbb{E}\{W(n)X^*(n-k)\}$$

For $k \geq 1$, the second member is equal to zero, since on one hand, $X(n-k)$ only depends on $W(n-k)$, $W(n-k-1)\ldots$ because of the stationary solution's causality, and on the other hand, $W(n)$ is white. If we use the stationarity of $X(n)$, and if we let $R(k) = \mathbb{E}\{X(n+k)X^*(n)\}$, then for any $k \geq 1$:

$$R(k) + a_1 R(k-1) + \cdots + a_P R(k-P) = 0 \tag{8.61}$$

We get the same relation as 8.59. Furthermore, if we multiply the two conjugate members of the recurrence relation 8.53 by $W(n)$ and if we consider the expectation, we obtain:

$$\mathbb{E}\{(X^*(n) + a_1 X^*(n-1) + \cdots + a_P X^*(n-P))W(n)\} = \mathbb{E}\{|W(n)|^2\} = \sigma^2$$

The first member is reduced to $\mathbb{E}\{X^*(n)W(n)\}$ because of the causality of $X(n)$ as a function of $W(n)$ and because $W(n)$ is white. As a consequence, we have $\mathbb{E}\{X^*(n)W(n)\} = \sigma^2$. Replacing $W(n)$ with $X(n) + a_1 X(n-1) + \cdots + a_P X(n-P)$ leads us to:

$$R(0) + a_1 R(-1) + \cdots + a_P R(-P) = \sigma^2 \tag{8.62}$$

If we stack 8.62 and the P relations we obtained from 8.61 for $k = 1, 2, \ldots, P$, we get 8.58.

You can recognize the $(P+1)$-th order covariance matrix of the process $X(n)$ in expression 8.58 . This matrix is hermitian in the general case, $R(-j) = R^*(j)$, and is symmetrical if the process $X(n)$ is real.

An important result states that, because the covariance matrix is a positive Toeplitz matrix, the solution to equation 8.58 is such that the polynomial $A(z) = 1 + a_1 z^{-1} + \cdots + a_P z^{-P}$ has all its roots inside the unit circle. This result is still true if the covariances are replaced with their estimates, so long as the matrix remains a positive Toeplitz matrix, as it is the case for the correlation method. The fact that $A(z)$ has all its roots inside the unit circle ensures that the filter with the transfer function $1/A(z)$ has a causal and stable representation. This filter is used for creating $X(n)$ from $W(n)$. We will see an application for it in speech processing.

Conversely, if we know the sequence $\{a_1, \ldots, a_P\}$ et σ^2, equation 8.58 allows us to calculate the covariance coefficients $R(k)$. The equations we have to solve are linear. Thus, in the case of a real random process for which $R(k) = R(-k)$, we get the expression:

$$\left(\begin{bmatrix} 1 & a_1 & \cdots & a_P \\ a_1 & & a_P & 0 \\ \vdots & \iddots & \iddots & \vdots \\ a_P & 0 & \cdots & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & \cdots & 0 \\ a_1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ a_P & \cdots & a_1 & 1 \end{bmatrix}\right) \begin{bmatrix} R(0)/2 \\ R(1) \\ \vdots \\ R(P) \end{bmatrix} = \begin{bmatrix} \sigma^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{8.63}$$

We have a system of $(P+1)$ linear equations with the unknowns $R(0)$, $\ldots$, $R(P)$. Once $R(0)$, $\ldots$,$R(P)$ have been calculated, 8.59 can be used to calculate the values of $R(k)$ beyond P, and the hermitian symmetry can be used for $k < 0$.

Example 8.10 (First order AR model)
Consider the first order, real AR process, solution of the equation $X(n) + a_1 X(n-1) = W(n)$ where a_1 is real with its modulus strictly less than 1, and $W(n)$ is a white, centered, WSS process with the variance σ^2.

1. Write the Yule-Walker equations.
2. Use them to find the covariances as a function of a_1 et σ^2.
3. Use this result to find a_1 and σ^2 as a function $R(0)$ of $R(1)$.

HINT:

1. The Yule-Walker equations are, for $P = 1$:

$$R(0) + a_1 R(1) = \sigma^2 \text{ and } R(1) + a_1 R(0) = 0 \tag{8.64}$$

   where we used $R(1) = R(-1)$. And for $k > 1$, equation 8.59 leads to $R(k) = -a_1 R(k-1)$.

2. By solving equations 8.64 with respect to $R(0)$ and $R(1)$, we get:

$$R(0) = \frac{\sigma^2}{1 - a_1^2}$$

   Using the recurrence relation 8.59 then, for $k < -1$, noticing that $R(k) = R(-k)$, leads us to:

$$R(k) = (-1)^k a_1^{|k|} \frac{\sigma^2}{1 - a_1^2} \tag{8.65}$$

3. We can also find a_1 and σ^2 from $R(0)$ and $R(1)$:

$$a_1 = -\frac{R(1)}{R(0)} \quad \text{and} \quad \sigma^2 = R(0) - \frac{R^2(1)}{R(0)} \tag{8.66}$$

These expressions can be used to estimate the parameters a_1 and σ^2 from a sequence of N observations. All we need to do is replace the covariance coefficients with their respective estimates (equation 8.32), and we get:

$$\widehat{a}_1 = -\frac{\sum_{n=0}^{N-2} X(n+1)X(n)}{\sum_{n=0}^{N-1} X^2(n)} \quad \text{and} \quad \widehat{\sigma}^2 = \frac{\sum_{n=0}^{N-1} X^2(n)}{N}(1-\widehat{a}_1^2)$$

■

Example 8.11 (Generating an exact Gaussian AR)
We have already seen how to generate a sequence of random processes with given spectra by filtering a white sequence. We discussed the problem regarding the transient state. In the case of a Gaussian AR process, we will see that an *exact* process trajectory can be generated. This can be used to initialize the previous method. Write a program:

- that picks M poles at random inside the unit circle and then use the MATLAB® function `poly` to calculate the coefficients of the polynomial using these poles and their conjugates;
- that calculates, using equations 8.63, the covariances of an AR process defined by $A(z)$ and σ^2;
- that generates $2M$ samples of the AR process. Use a method similar to that of example 8.5 on page 283 where we generated a colored noise using a white noise and the square root of its covariance matrix;
- that constructs a trajectory with the length T of this process from the $2M$ previous samples. Use the `filtricII` function to calculate, as a function of the $2M$ previous samples, the *initial state* that must be given to the `filter` function;
- that plots the sequence of covariance;
- that plots in 2D the couples $\{x_i, x_{i+1}\}$.

HINT: type:

```
%===== EXACTAR.M
% Polynomial generation
sigma2=2; M=5; rho=0.45*rand(1,M)+0.5; phi=pi*rand(1,M)/4;
```

```
rac=rho .* exp(j*phi); rac=[rac conj(rac)];
coeff=real(poly(rac)); LgAR=length(coeff)-1;
%===== Calculation of the exact covariances
c1=[coeff zeros(1,LgAR+1)]; c2=[zeros(1,LgAR) coeff];
A1=toeplitz([coeff(LgAR+1);zeros(LgAR,1)],coeff(LgAR+1:-1:1));
A1=A1(:,LgAR+1:-1:1);
A2=toeplitz(coeff,[coeff(1);zeros(LgAR,1)]);
rx=(A1+A2)\[sigma2; zeros(LgAR,1)]; rx(1)=rx(1)*2;
Rcov=toeplitz(rx);
%===== Square root of the covariance matrix
MatM=sqrtm(Rcov(1:LgAR,1:LgAR));
%===== Generating the first Ncoeff-1 values of X(k)
W0=randn(LgAR,1); X0=MatM*W0; Z0=filtricII(1,coeff,0,X0);
T=200; W=sqrt(sigma2)*randn(T,1);
Xf=filter(1,coeff,W,Z0); LXf=length(Xf);
Xf=Xf-mean(Xf); R=2*LgAR; covX=zeros(R,1);
for rr=1:R, covX(rr)=Xf(rr:T)'*Xf(1:T-rr+1)/T; end
subplot(221); mycirc=exp(2*j*pi*(0:100)/100); plot(mycirc);
hold on; plot(rac,'x'); hold off; axis('square'); grid
subplot(222); plot(Xf); grid
subplot(223); stem(covX/covX(1));
subplot(224); plot(Xf(1:LXf-1),Xf(2:LXf),'.'); grid
```

A high correlation between consecutive samples corresponds to poles near the unit circle and gives a 2D-plotting in which points are gathered around a straight line. ■

8.5.3 The Levinson algorithm

We have seen that the covariance matrix of a WSS process is a Toeplitz matrix. We are now going to give a fast algorithm, originally suggested by Levinson, for solving the Yule-Walker equations 8.58. If k is the size of the covariance matrix, this algorithm has a complexity in k^2 whereas a general inversion algorithm is in k^3. It is useful only when implemented in a language like "C", or "Fortran", since MATLAB® provides a *built-in* type matrix inversion function (the `inv` function). In the most recent versions, the *signal toolbox* has a `levinson` function.

The Levinson algorithm is recursive. It calculates the coefficients of the m-th *step* coefficients using those obtained in the $(m-1)$-th step. To achieve this recursion, we choose the following notation for equation 8.58, in which we used the hermitian symmetry $R(-k) = R^*(k)$:

$$\begin{bmatrix} R(0) & R^*(1) & \cdots & R^*(m-1) \\ R(1) & R(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & R^*(1) \\ R(m-1) & \cdots & R(1) & R(0) \end{bmatrix} \begin{bmatrix} a_{m-1}(0) \\ a_{m-1}(1) \\ \vdots \\ a_{m-1}(m-1) \end{bmatrix} = \begin{bmatrix} v_{m-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{8.67}$$

In this expression, the index $(m-1)$ indicates the $(m-1)$-th step solution, $a_{m-1}(0)=1$ and v_{m-1} refers to the variance of the input process. We will see in section 11.4 that the coefficients $a_{m-1}(k)$ have a fundamental meaning in the linear prediction problem.

We can now rewrite expression 8.67 in the following two ways:

$$\mathbf{R}_{m-1}\mathbf{a}_{m-1}^{F}=\begin{bmatrix} v_{m-1}\\ \mathbf{0}_{m-2}\end{bmatrix} \text{ and } \mathbf{R}_{m-1}\mathbf{a}_{m-1}^{B*}=\begin{bmatrix} \mathbf{0}_{m-2}\\ v_{m-1}^{*}\end{bmatrix}$$

where we have assumed[6]:

$$\begin{aligned}\mathbf{a}_{m-1}^{F} &= [1 \quad a_{m-1}(1) \quad \cdots \quad a_{m-1}(m-1)]^{T}\\ \mathbf{a}_{m-1}^{B} &= [a_{m-1}(m-1) \quad \cdots \quad a_{m-1}(1) \quad 1]^{T}\end{aligned}$$

The exponents F and B (as in *forward* and *backward*) indicate the direction chosen for the vectors.

By going on to the m-th step, the covariance matrix can be written:

$$\mathbf{R}_{m}=\begin{bmatrix} \mathbf{R}_{m-1} & \mathbf{r}_{m}^{B*}\\ \mathbf{r}_{m}^{BT} & R(0)\end{bmatrix}=\begin{bmatrix} R(0) & \mathbf{r}_{m}^{FH}\\ \mathbf{r}_{m}^{F} & \mathbf{R}_{m-1}\end{bmatrix}$$

where $\mathbf{r}_{m}^{B}=[R(m) \quad \cdots \quad R(1)]^{T}$ and $\mathbf{r}_{m}^{F}=[R(1) \quad \cdots \quad R(m)]^{T}$. Using the expression of the $(m-1)$-th step solution, we then get:

$$\mathbf{R}_{m}\begin{bmatrix} \mathbf{a}_{m-1}^{F}\\ 0\end{bmatrix}=\begin{bmatrix} \mathbf{R}_{m-1} & \mathbf{r}_{m}^{B*}\\ \mathbf{r}_{m}^{BT} & R(0)\end{bmatrix}\begin{bmatrix} \mathbf{a}_{m-1}^{F}\\ 0\end{bmatrix}=\begin{bmatrix} \begin{bmatrix} v_{m-1}\\ \mathbf{0}_{m-2}\end{bmatrix}\\ \mathbf{r}_{m}^{BT}\mathbf{a}_{m-1}^{F}\end{bmatrix}$$

$$\mathbf{R}_{m}\begin{bmatrix} 0\\ \mathbf{a}_{m-1}^{B*}\end{bmatrix}=\begin{bmatrix} R(0) & \mathbf{r}_{m}^{FH}\\ \mathbf{r}_{m}^{F} & \mathbf{R}_{m-1}\end{bmatrix}\begin{bmatrix} 0\\ \mathbf{a}_{m-1}^{B*}\end{bmatrix}=\begin{bmatrix} \mathbf{r}_{m}^{FH}\mathbf{a}_{m-1}^{B*}\\ \begin{bmatrix} \mathbf{0}_{m-2}\\ v_{m-1}^{*}\end{bmatrix}\end{bmatrix}$$

By linear combination:

$$\mathbf{R}_{m}\left[\begin{bmatrix} \mathbf{a}_{m-1}^{F}\\ 0\end{bmatrix}+k_{m}\begin{bmatrix} 0\\ \mathbf{a}_{m-1}^{B*}\end{bmatrix}\right]=\begin{bmatrix} v_{m-1}+k_{m}\mathbf{r}_{m}^{FH}\mathbf{a}_{m-1}^{B*}\\ \mathbf{0}_{m-2}\\ \mathbf{r}_{m}^{BT}\mathbf{a}_{m-1}^{F}+k_{m}v_{m-1}^{*}\end{bmatrix} \tag{8.68}$$

By choosing:

$$k_{m}=-\frac{\mathbf{r}_{m}^{BT}\mathbf{a}_{m-1}^{F}}{v_{m-1}^{*}} \tag{8.69}$$

the last term is set to zero and, by identification, we obtain the m-th step solution, which leads to $v_{m}=v_{m-1}+k_{m}\mathbf{r}_{m}^{FH}\mathbf{a}_{m-1}^{B*}$. By noticing that by definition, $\mathbf{r}_{m}^{FH}\mathbf{a}_{m-1}^{B*}=(\mathbf{r}_{m}^{BT}\mathbf{a}_{m-1}^{F})^{*}$, we infer from 8.69 that

$$\mathbf{r}_{m}^{FH}\mathbf{a}_{m-1}^{B*}=-k_{m}^{*}v_{m-1}$$

[6] Remember that the exponent ^{T} refers to transposition *without conjugation*.

and therefore that:

$$v_m = v_{m-1}(1 - |k_m|^2) \tag{8.70}$$

Because $v_m \geq 0$ for any m, then $(1 - |k_m|^2) \geq 0$ and $0 \leq v_m \leq v_{m-1}$ and:

$$|k_m| \leq 1 \tag{8.71}$$

The coefficients k_m are called the *reflection coefficients.*

By identification, we infer from 8.68 the m-th step coefficients as functions of those obtained at the $(m-1)$-th step:

$$\mathbf{a}_m^F = \begin{bmatrix} \mathbf{a}_{m-1}^F \\ 0 \end{bmatrix} + k_m \begin{bmatrix} 0 \\ \mathbf{a}_{m-1}^{B*} \end{bmatrix} \tag{8.72}$$

In particular, $a_m(0) = 1$ and $a_m(m) = k_m$. To sum up, starting off with the sequence of the covariances $R(k)$, or of their estimates in practice, the Levinson algorithm can be written as follows:

```
Initial values: $a_0(0) = 1$ and $v_0 = R(0)$
For $m = 1, \ldots, K$, repeat:
    1. $k_m = -\frac{R(m)a_{m-1}(0) + \cdots + R(1)a_{m-1}(m-1)}{v_{m-1}}$
    2. $a_m(0) = 1, a_m(m) = k_m$
    3. For $j \in \{1, \cdots, m-1\} : a_m(j) = a_{m-1}(j) + k_m a_{m-1}^*(m-j)$
    4. $v_m = v_{m-1}(1 - |k_m|^2)$
```

We have to check that for the step $m = 1$, we have:

$$k_1 = -R(1)/R(0), \quad a_1(0) = 1, \quad a_1(1) = k_1, \quad v_1 = v_0(1 - |k_1|^2)$$

In the case of an AR-P process, we will show that the coefficients $a_m(m) = 0$ for $m \geq P + 1$, which makes it possible to stop the previous loop.

Exercise 8.6 (Levinson algorithm)

1. Write a function that calculates the covariance estimates then the parameters of a P order AR model using the Levinson algorithm. Check the function on a P order AR model by comparing with the estimates from the Yule-Walker inversion. Check that $a_m(m) = 0$ when $m \geq P+1$. The Levinson algorithm also returns the reflection coefficients k_m.

2. Using a structure similar to the one used in the Levinson algorithm for solving the matrix equation of the type $\mathbf{R}[1\ a_1\ \cdots\ a_p]^T = [\sigma^2\ 0 \cdots 0]^T$, imaginer an algorithm that can solve the equation $\mathbf{Rh} = \mathbf{c}$, where $\mathbf{R}$ is a Toeplitz matrix and $\mathbf{c}$ is any vector. We will run into this type of equations in section 11.2.5 when identifying a filter: see for example equations 11.22 and 11.42 from Chapter 11. $\mathbf{R}$ is an autocovariance matrix, $\mathbf{c}$ is a covariance vector between the input and the output and $\mathbf{h}$ is an FIR filter we have to estimate.

8.5.4 ARMA (P, Q) process

ARMA processes are obtained using an AR structure and an MA structure in series. The process is the solution to the recursive equation:

$$X(n)+a_1X(n-1)+\cdots+a_PX(n-P) = W(n)+b_1W(n-1)+\cdots+b_QW(n-Q) \tag{8.73}$$

where $W(n)$ refers to second order stationary, centered, white random process with the variance σ^2 and where $\{a_1, \ldots, a_P\}$ and $\{b_1, \ldots, b_Q\}$ are two sequences of coefficients. Let:

$$H_z(z) = \frac{1 + b_1 z^{-1} + \cdots + b_Q z^{-Q}}{1 + a_1 z^{-1} + \cdots + a_P z^{-P}} \tag{8.74}$$

It can be proven that equation 8.73 has a single, second-order stationary solution $X(n)$ if and only if the denominator's roots, that is the poles of the transfer function $H_z(z)$, have a modulus different from 1. In the case where $A(z) \neq 0$ for $|z| \geq 1$, the poles of $H_z(z)$ are strictly inside the unit circle, $X(n)$ can be causally expressed as a function of $W(n)$.

By definition, an ARMA (P, Q) process is the stationary solution to the recursive equation of the type 8.73.

Spectrum of an ARMA-(P, Q)

Because an ARMA process can be interpreted as the output of a filter with a white noise with the PSD σ^2 as its input, we can use formula 8.39 to determine an expression of the PSD. This leads us to:

$$S(f) = \sigma^2 \frac{\left|1 + b_1 e^{-2j\pi f} + \cdots + b_Q e^{-2j\pi Q f}\right|^2}{\left|1 + a_1 e^{-2j\pi f} + \cdots + a_P e^{-2j\pi P f}\right|^2} \tag{8.75}$$

Comments

Using an ARMA process as a model for describing an observation amounts to assuming that its spectrum is a rational function. This is why it can seem restrictive to assume that the second order parameters of an observation only depend on a finite number of parameters. From the engineer's perspective, the great "universal" nature of this model is due to the fact that rational transfer functions make it possible to approximate a very large number of functions, and quite naturally play a role in electric, electronic, or even mechanical devices, often by way of a constant coefficient linear recursive equation.

Furthermore, any ARMA of MA process can be approximated with an AR order of a high enough order. This result is fundamental for practical applications, since if among these three models the wrong one is chosen, a reasonable accuracy can still be achieved by taking a high enough order. However, it is easily conceivable that a process for which the spectrum has "deep valleys" will require less parameters if the MA model is used to represent it rather than an AR model. And conversely for "high peaks".

Finally, remember that estimating the coefficients of an MA model usually is not simple, since the relations 8.49 between the model's coefficients and the covariances are not linear, whereas they are for an AR model. This is why we will only be estimating the parameters of an AR model.

Chapter 9

Continuous Spectra Estimation

The object of this chapter is mainly a discussion of the power spectral density's (PSD) estimation. In this field, it is customary to separate two cases:

- When the statistical properties of the observation depend on a finite, and usually small number of parameters, the model is said to be *parametric.* To be more precise, it means that knowing the few useful parameters is enough to find the exact probability distribution of the observation. We have already encountered an example of the parametric model: the AR-P model in the case of a white Gaussian input. Knowing the $(P+1)$ parameters a_1, ..., a_P, σ^2 is enough to determine the probability distribution. Without the Gaussian hypothesis, and although it is no longer possible to write the probability distribution of the observation precisely, it is still possible to estimate some useful quantities, such as its spectrum, given a finite number of parameters: this is sometimes called a *semiparametric* model.

- Otherwise, the model is said to be *non-parametric.* In the first paragraph of this chapter we will study a situation in which the only hypothesis is that the process is WSS. Knowing the spectrum requires the estimation of an infinity of parameters, that is the set of covariance coefficients.

9.1 Non-parametric estimation of the PSD

9.1.1 Estimation from the autocovariance function

We have seen that expressions 8.32 can be used to estimate $R(k)$ from a series of N observations $X(1)$, ..., $X(N)$. Using the resulting estimate of the autocovariance function and the definition 8.13 of the PSD, an estimate of the latter can be obtained by:

$$\widehat{S}(f) = \sum_{k=-(K-1)}^{K-1} \widehat{R}(k)e^{-2j\pi fk} \tag{9.1}$$

For a *real* WSS random process, we can write 9.1 as follows:

$$\widehat{S}(f) = \widehat{R}(0) + 2\sum_{k=1}^{K-1} \widehat{R}(k)\cos(2\pi kf) \tag{9.2}$$

since $R(k) = R(-k)$.

COMMENTS:

- Even if we have at our disposal the actual values of the autocovariance function (assumed to be with infinite support), the fact of restricting, in the calculation of $S(f)$, the sequence $R(k)$ to $k \in \{-(K-1), \dots, (K-1)\}$ amounts to multiplying $\{R(k)\}$ by the rectangular window $w_r(k) = \mathbb{1}(k \in \{-(K-1), \dots, (K-1)\})$. This operation causes unwanted ripples, as we saw in Chapters 3 and 4. Because the lobes can have positive or negative values, this can result in negative values for the PSD estimate. Losing the positive nature (see page 279) of the PSD is not advisable. To avoid such a phenomenon, the *triangular* window, or *Bartlett* window can be used instead of the rectangular window, the expression of which is:

  $$w_b(k) = \left(1 - \frac{|k|}{K}\right)\mathbb{1}(k \in \{-(K-1), \dots, (K-1)\}) \tag{9.3}$$

  It ensures the positive nature of the result, because the triangle function is obtained by convolution of the rectangular window with itself. The DTFT of the sequence $w_b(k)$ is therefore the function $\sin^2(\pi(2K-1)f)/\sin^2(\pi f)$, which is always positive. Another commonly used window is the *Hamming* window, the expression of which, as you may remember, is:

  $$w_h(k) = \left[0.54 + 0.46\cos\left(\frac{\pi k}{K-1}\right)\right]\mathbb{1}(k \in \{-(K-1), \dots, (K-1)\}) \tag{9.4}$$

- Another important element is the choice of the number K of estimated covariance points compared with the number N of observations. Consider the case where $N \to \infty$ (large samples). If K remains constant, the covariance coefficients (expression 8.29) become more and more precise, but the windowing effect remains. Hence the idea of increasing K as N increases, but much "slower" than N. For example, we can take $K = \lambda N^\alpha$ with $\lambda \approx 1/10$ and $\alpha < 1$. That way, when N tends to infinity, a number K, that tends to infinity, of covariance points are calculated while ensuring that the number of values used for estimating each covariance point also tends to infinity.

The following function estimates the spectrum by DFT of K covariance coefficients estimated from N observations:

```
function sf=covtodsp(x,K,wintype,Lfft)
%%====================================================%
%% Estimating the spectrum from the covariances       %
%% SYNOPSIS: sf=COVTODSP(x,K,wintype,Lfft)            %
%%    x       = Input sequence                        %
%%    K       = Number of estimated covariances       %
%%    wintype = 'r', 'h' ou 'b'                       %
%%         for 'rectangular', 'hamming' or 'bartlett' %
%%    Lfft    = FFT size                              %
%%    sf      = PSD                                   %
%%====================================================%
N=length(x); x=x(:); rx=zeros(1,K); x=x-ones(1,N)*x/N;
%===== Estimating the K covariances
for ii=1:K, rx(ii)=x(1:N-ii+1)'*x(ii:N); end
rx=rx/N;
%===== Windowing
if wintype(1) == 'b',
      rx=rx .* (K:-1:1)/K;
   elseif wintype(1) == 'h'
      rx=rx .* (0.54+0.46*cos(pi*(0:K-1)/K));
end
%===== Using the hermitian symmetry property
rx(1)=rx(1)/2; sf=fft(rx,Lfft); sf=2*real(sf);
return
```

The program is designed to process complex signals. In this case, the spectrum is of course real and positive, but no longer has the even symmetry anymore. Any of the rectangular, Hamming or Bartlett, windows may be used by assigning to `fentype` one of the three values `'r'`, `'h'` or `'b'`. Calculating the PSD uses the hermitian symmetry property of the covariance function: remember (see exercise 2.2 in Chapter 2) that to calculate the DFT of the sequence, completed by hermitian symmetry, all you have to do is take the real part multiplied by 2 of the monolateral sequence after having divided the first element by 2. The behaviors for a Bartlett window are illustrated by the spectra represented in dB in Figures 9.1 and 9.2 for two values of the number of covariance estimates, $K = 25$ and $K = 150$. They are obtained using the program:

```
%===== TESTCOVTODSP.M
N=500; Lfft=1024; freq=(0:Lfft-1)/Lfft;
%===== Complex process generated by the all-pole filtering
```

```
%      of a white noise
a=[1 -2.4788 3.0905 -2.0646 0.6856];
sw=[1;0*j]; Pw=sw'*sw;
w=randn(N,2)*sw; x=filter(1,a,w);
%===== K and the window type can be modified
K=25; sest=covtodsp(x,K,'b',Lfft);
sestlog=10*log10(sest);
sth =10*log10(Pw * abs(1 ./ fft(a,Lfft)) .^2);
plot(freq,sth,freq,sestlog,'r'); grid
set(gca,'xlim',[0 1/2]); % if x is real
```

The real signal x is generated by a white noise filtered with $N = 500$ values. The theoretical spectrum is given by the filtering formula, equation 8.39.

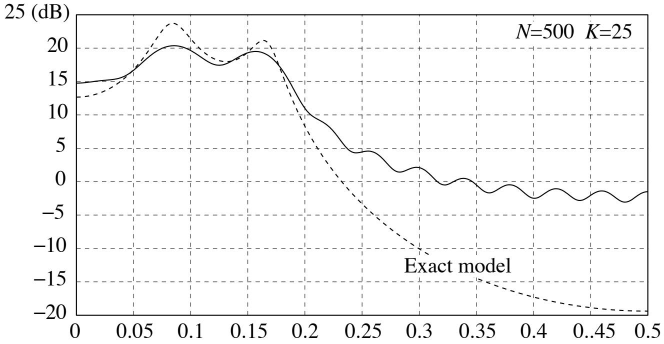


Figure 9.1 – *Non-parametric estimation of an AR-4 spectrum based on $K = 25$ estimates of the covariance coefficients. The length of the sample is $N = 500$. The window used is the Bartlett window. The dashed line is the theoretical PSD*

As you can see, when the number of estimates of the covariance coefficients increases, the estimated spectrum's fluctuations are "closer" to the theoretical spectrum, but the fluctuations have higher amplitudes. You can easily check with the program that the use of the rectangular or Hamming windows does not ensure that the estimated PSD is positive (if not, the program returns an error message when the command `plot(freq,sth,freq,sest,'r')` is executed, since `sest` contains complex numbers because of the fact that the logarithm of a negative number is a complex number). A complex signal can be used by changing for example `sw=[1;0*j]` to `sw=[1;3*j]` and suppressing the last line.

9.1.2 Estimation based on the periodogram

Rather than to use the DTFT of the covariances to estimate the PSD, an intuitive idea would be to start off with the Fourier transform of a trajectory, or of a portion of a trajectory and to calculate its square modulus. This leads to the following definition of the *periodogram*.

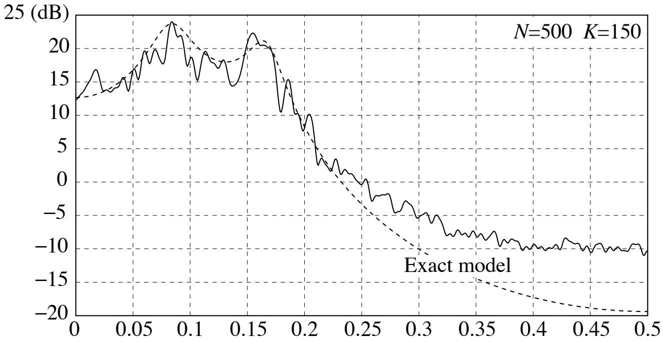


Figure 9.2 – *Non-parametric estimation of an AR-4 spectrum based on $K = 150$ estimates of the covariance coefficients. The length of the sample is $N = 500$. The window used is the Bartlett window*

Definition 9.1 (Periodogram) *Let $X(n)$ be a centered WSS random process. A periodogram is the random function of $f \in (0,1)$ defined by:*

$$I_N(f) = \frac{1}{N}\left|\sum_{n=0}^{N-1} X(n)e^{-2j\pi nf}\right|^2 \tag{9.5}$$

You would think that $I_N(f)$ might be a good estimator of the PSD $S(f)$ of the process $X(n)$ assumed to be WSS, but in fact *not at all.* Although the mean $\mathbb{E}\{I_N(f)\}$ tends to the "true" value $S(f)$ when N tends to infinity, the square deviation $\mathbb{E}\left\{|I_N(f) - S(f)|^2\right\}$ does not tend to zero when N tends to infinity.

We will now prove that the mathematical expectation of $I_N(f)$ tends to $S(f)$. We have:

$$\begin{aligned}\mathbb{E}\{I_N(f)\} &= \frac{1}{N}\mathbb{E}\left\{\sum_{n=0}^{N-1} X(n)e^{-2j\pi fn}\sum_{m=0}^{N-1} X^*(m)e^{2j\pi fm}\right\}\\ &= \frac{1}{N}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} R(n-m)e^{-2j\pi f(n-m)}\\ &= \sum_{k=-(N-1)}^{N-1}\left(1-\frac{|k|}{N}\right)R(k)e^{-2j\pi fk}\end{aligned}$$

where we used $\mathbb{E}\{X(n)X^*(m)\} = R(n-m)$ (the process is assumed to be WSS and centered) and then the identity:

$$\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} g(n-m) = \sum_{k=-(N-1)}^{N-1}\left(1-\frac{|k|}{N}\right)g(k) \tag{9.6}$$

We can rewrite $\mathbb{E}\{I_N(f)\}$ as follows:

$$\begin{aligned}\mathbb{E}\{I_N(f)\} &= \sum_{k=-\infty}^{+\infty}\left(1-\frac{|k|}{N}\right)\mathbb{1}(k\in\{-(N-1),\dots,(N-1)\})R(k)e^{-2j\pi fk}\\ &= \sum_{k=-\infty}^{+\infty} f_N(k)R(k)e^{-2j\pi fk}\end{aligned}$$

If we assume that $R(k)$ is summable, we can calculate the limit of $\mathbb{E}\{I_N(f)\}$ by swapping the limit and the sum sign (dominated convergence theorem [28]). By noticing that $f_N(n)$ tends to 1, we infer that $\mathbb{E}\{I_N(f)\}$ tends to $S(f)$ when N tends to infinity. Hence the periodogram is an *asymptotic unbiased estimator* of the PSD: for N high enough, $I_N(f)$ fluctuates around the "true" value $S(f)$.

However, it can be proven, and we will assume so, that the amplitude of the fluctuations, that is $\mathbb{E}\left\{|I_N(f)-S(f)|^2\right\}$, does not tend to 0 when N tends to infinity. To be more precise, what can be shown is that, under very general hypotheses, this quantity can have the same order of magnitude as the value we wish to estimate.

The periodogram fluctuates around the true PSD. Even if N is very high, the amplitude of the fluctuations still has the same order of magnitude as the PSD we wish to estimate.

The following program illustrates this behavior:

```
%===== FLUCTPERIO.M
Lfft=1024; fq=(0:Lfft-1)/Lfft;
w=randn(1,1000); b=[1 1.2 0.9]; a=[1 -1.1 0.92];
PSDth= 20*log10( abs(fft(b,Lfft) ./ fft(a,Lfft)));
x=filter(b,a,w);            %===== Process
lxt=[100 200 500 1000];
for ii=1:length(lxt)
    xt=x(1:lxt(ii));
    per=20*log10(abs(fft(xt,Lfft))-10*log10(lxt(ii));
    subplot(2,2,ii); plot(fq,per,fq,PSDth,'w'); grid
    axis([0 .5 -30 40]);
    title(sprintf('N = %d',lxt(ii)));
end
```

The samples of the random process $X(n)$ are obtained by filtering white noise with a variance of 1. Based on the transfer function of the filter $H_z(z) = (1+1.2z^{-1}+0.9z^{-2})/(1-1.1z^{-1}+0.92z^{-2})$, the expression of the theoretical PSD of the process $X(n)$ is:

$$S(f) = \frac{|1+1.2e^{-2j\pi f}+0.9e^{-4j\pi f}|^2}{|1-1.1e^{-2j\pi f}+0.92e^{-4j\pi f}|^2}$$

Figure 9.3 shows the periodograms for four values of N as well as the theoretical PSD. The periodograms fluctuate around the exact PSD and the amplitude of these fluctuations does not seem to decrease when N increases.

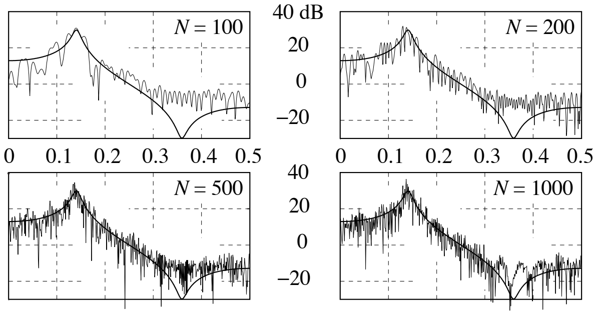


Figure 9.3 – *Fluctuations of the periodogram for several values of N*

Therefore, there is no point in directly using the periodogram for estimating the spectrum. However, in practice, every good estimator of the PSD is constructed from the periodogram. We are going to explain two methods: the smoothed periodogram and the averaged periodogram.

Frequency smoothed periodogram

The periodograms in Figure 9.3 show that the values obtained in several neighboring frequency points fluctuate around the actual value: some are very close, whereas others are very far, hence the idea of calculating a frequency mean. To be more precise, consider an integer M and a sequence $W_{M,N}(k)$ of weighting coefficients such that:

1. for any k, $W_{M,N}(k) = W_{M,N}(-k)$ and $W_{M,N}(k) \geq 0$;
2. $\sum_{|k|\leq M} W_{M,N}(k) = 1$;
3. $\sum_{|k|\leq M} W^2_{M,N}(k) \to 0$ when $N \to \infty$;
4. $\left[\sum_{|k|\leq M} k^2 W_{M,N}(k)\right] / N^2 \to 0$ when $N \to \infty$.

If a relation of the type:

$$M = N^\alpha \text{ with } \alpha < 1 \tag{9.7}$$

is chosen, then the above conditions are met. This is particularly the case for the rectangular window defined by:

$$W_{M,N}(k) = \frac{1}{2M+1}\mathbb{1}(k \in \{-M, \cdots, M\})$$

and of the triangular window, or Bartlett window, defined by:

$$W_{M,N}(k)=\frac{1}{M}\left(1-\frac{|k|}{M}\right)\mathbb{1}(k\in\{-M+1,\cdots,M-1\})$$

In the case of a rectangular window, condition 3 can be expressed:

$$\sum_{|k|\leq M} W^2_{M,N}(k) \quad = \quad \sum_{|k|\leq M}\frac{1}{(2M+1)^2}=\frac{1}{2M+1}$$

If M tends to infinity with N, then this condition 3 is satisfied. Condition 4 can be expressed:

$$\begin{aligned}\frac{\sum_{|k|\leq M} k^2 W_{M,N}(k)}{N^2} &= \sum_{|k|\leq M} k^2\frac{1}{(2M+1)N^2}\\ &= \frac{1}{N^2(2M+1)}\sum_{|k|\leq M} k^2=\frac{M(M+1)}{3N^2}\end{aligned}$$

The condition is satisfied if $M/N\to 0$ at infinity, which is the case with the conditions set forth.

Then we calculate the periodogram of the sequence $\{X(0),\ldots,\ X(N-1)\}$ at the points of frequencies k/N. Finally, the spectrum is estimated according to the expression:

$$\widehat{S}_N(m/N)=\sum_{k=-M}^{M} W_{M,N}(k)I_N((m+k)/N) \tag{9.8}$$

To deal with the side effects, remember that $I_N(f)$ is periodic with period 1.

Obviously, expression 9.8 leads to a decrease in the variance but on the other hand adds bias. A detailed study of the properties of various window and of the compromise between bias and variance can be found in [14]. We will only be considering the triangular window.

Example 9.1 (Smoothed periodogram) Write a function that smooths the periodogram using either a rectangular window or a triangular window, with a length of $2m+1$ and with the sum of its coefficients equal to 1. Making the choice of $L=N$ avoids withdrawing the empirical mean since $\sum_{n=0}^{N-1}\mu e^{-2j\pi kn/N}=0$ for all $k\neq 0$ mod N. To make the choice of the window's length automatic, you can take $M=N^{2/5}$ where N is the length of the signal.

HINT: type:

```
function [sf,frq]=smperio(x,M,window)
%%============================================%
%% Smoothed periodogram                       %
%% SYNOPSIS: [sf,frq]=SMPERIO(x,M, window)    %
%%     x      = Input sequence                    %
%%     M such that the window length is 2*M+1     %
%%     window = 'r' for rectangular               %
%%            = 't' for triangular                %
%%     frq    = Frequencies for the estimated PSD %
%%     sf     = Spectrum                          %
%%================================================%
if nargin<3, window='t'; end
x=x(:); N=length(x);
if nargin<2, M=N^(2/5); end
sf=zeros(N-2*M+1,1);
if window=='t'
      Wf=[(1:M+1) (M:-1:1)]/(M+1)^2;
else
      Wf=ones(1,2*M+1)/(2*M+1);
end
Periodogram=abs(fft(x)) .^2/N; % Periodogram
frq=(M+1:N-M-1)/N; sf=filter(Wf,1,Periodogram); sf=sf(2*M+2:N);
```

■

Averaged periodogram

This method, suggested by P. Welch [101], consists of cutting up the signal in blocks and to calculate a mean of the different periodograms obtained for each block. To be more explicit, the algorithm is as follows:

1. *The sample $\{X(n)\}$ of length N is divided in L blocks of the same length K with overlapping.*
2. *A weighting window is applied to each of the L blocks. The resulting sequences are denoted by $\{\widetilde{X}_\ell(k)\}$ with $\ell = 0 : L-1$ and $k = 0 : K-1$.*
3. *The L periodograms are calculated, as well as their mean:*

$$\widehat{S}_N(f) = \frac{1}{L}\sum_{l=0}^{L-1}\left(\frac{1}{K}\left|\sum_{k=0}^{K-1} \widetilde{X}_\ell(k)e^{-2j\pi kf}\right|^2\right) \tag{9.9}$$

This algorithm calls for a few comments:

1. Numerically, 9.9 is calculated on a finite number of values of $f = m/M$ and $m \in \{0, \ldots, M-1\}$ using the `fft` function of MATLAB®.

2. It can be shown, with relatively general conditions, that $\widehat{S}_N(f)$ is a spectrum estimator the variance of which tends to 0 when N tends to infinity. One condition in particular is that L also has to tend to infinity, but not as fast as N. To make the choice of L automatic, you can take for example $L = N^{1/3}$ or $L = N^{2/5}$.

3. In practice, the variance is reduced by choosing a large value for L. However, for a given value of N, increasing the number L of periodograms amounts to reducing the number of points $K = N/L$ of each analysis window and at the same time to reducing the frequency resolution. You may remember that this is because when a signal is observed over a duration of K, you cannot "see" frequency differences of less than $1/K$. The consequence is that the various periodograms show a discrepancy with the actual spectrum: this is called a bias. The conclusion is that choosing L is a compromise between bias and variance.

4. The absence of weighting in expression 9.9 amounts to multiplying the samples $X(k)$ of the signal by the rectangular window:

   $$w_r(k) = \frac{1}{\sqrt{K}}\mathbb{1}(k \in \{0, \ldots, (K-1)\})$$

   the energy of which is $\sum_k w_r^2(k) = 1$. This causes unwanted ripples to appear, related to the side lobes of its DTFT, hence the idea of using a different window to reduce these ripples. The consequence, of course, is a decrease in the resolution related to the main lobe. Generally speaking, the results are the same as those already found when studying the spectral analysis of deterministic signals, or filter design:

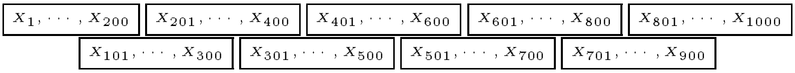
   – the wider the main lobe is, the more the details of the spectrum will be rubbed out;
   – the higher the second lobes are, the stronger the induced ripples will be. This is particularly noticeable in the areas of the spectrum showing few variations.

   The Hamming window is very often used:

   $$w_h(k) = \alpha\,(0.54 - 0.46\cos(2\pi k/K))\,\mathbb{1}(k \in \{0, \cdots, (K-1)\})$$

   where α is chosen such that $\sum_k w_h^2(k) = 1$.

5. Multiplying some of the samples by very small weighting coefficients (about 0.08 for the smallest values of the Hamming window) gives these samples a very unimportant role in the calculation. This is why *Welch* had the idea of choosing the intervals so they would overlap. The most commonly used overlap factor is 50%. For example, if $N = 1{,}000$ and $K = 200$, we get the following intervals:


$x_1, \cdots, x_{200}$
$x_{201}, \cdots, x_{400}$
$x_{401}, \cdots, x_{600}$
$x_{601}, \cdots, x_{800}$
$x_{801}, \cdots, x_{1000}$
$x_{101}, \cdots, x_{300}$
$x_{301}, \cdots, x_{500}$
$x_{501}, \cdots, x_{700}$
$x_{701}, \cdots, x_{900}$

Each sub-interval is then weighted by the appropriate Hamming window. The 9 periodograms, and then the mean are calculated.

6. Averaging the periodograms does not completely eliminate the "fluctuations". To be more specific the reference [54] gives a $\beta\%$ confidence interval involving what is called the χ-square distribution. If the latter is approximated by a Gaussian distribution, we infer that the spectrum has a $\beta\%$ chance of being in the interval:

$$\left[\frac{\hat{S}_w(f)}{1+\gamma} \quad \frac{\hat{S}_w(f)}{1-\gamma}\right] \quad \text{with} \quad \gamma = \frac{\sqrt{2}\text{erfinv}(\beta)}{\sqrt{L}} \tag{9.10}$$

where $\hat{S}_w(f)$ is the periodogram averaged using the Welch method (50% overlap). The erfinv(β) function is the inverse function of the error function, which can be called in MATLAB® using the command `erfinv`. For $\beta = 95\%$ and $L = 30$, we get, in decibels, the confidence interval $(\hat{S}_w(f) - 1.33\ ,\ \hat{S}_w(f) + 1.92)$.

Exercise 9.1 (Estimating the spectrum using the Welch method)

1. Write a MATLAB® function that estimates the spectrum using the Welch method, with as the input the signal to be analyzed, the type of window (rectangular or Hamming), and the number of points of the spectra.

2. Using the `filter` function, generate a signal corresponding to a given spectrum. Use the previous function to estimate the PSD of this test signal. Compare the result with the spectrum obtained with the `smperio.m` function of example 9.1. Compare it with the theoretical spectrum.

Exercise 9.2 (Estimating the spectrum of a binary signal)

Write a MATLAB® function that associates with each term of the sequence a_n of independent random variables with possible values -1 or 1, either the signal $g(n)$ or the signal $-g(n)$ respectively. The signal $g(n)$ is a rectangular impulse made up of a sequence of $T_1 = 10$ values equal to $A = 5$, followed by a sequence of $T_2 = 25$ values equal to 0. Figure 9.4 shows this signal for a sequence of 7 successive values of a_n. This signal could originate from the sampling of a "computer" signal used for transmitting a sequence of bits.

Use the `welch.m` function to estimate the spectrum of the signal associated with a sequence of 1024 values. Compare with the theoretical spectrum the expression of which, according to formula 12.55, is given by:

$$S(f) = \frac{1}{T}\left|G(f)\right|^2 = \frac{1}{T}\left|\frac{\sin(\pi f T_1)}{\sin(\pi f)}\right|^2 \tag{9.11}$$

where $T = T_1 + T_2$ is the duration of the sample and $G(f)$ is the DTFT of the sequence $g(n)$. $S(f)$ is obtained either by applying the `fft` function to the sequence that defines $g(n)$, or by using expression 9.11.

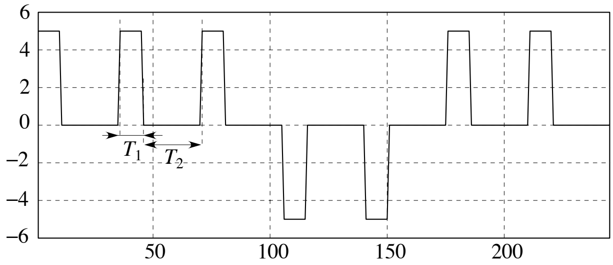


Figure 9.4 – *Binary signal. The positive impulse corresponds to the bit* 1 *and the negative impulse to the bit* 0*. Each impulse is comprised of a constant amplitude for a duration of* 10 *followed by a zero amplitude for a duration of* 25

Example 9.2 (Estimating the PSD of the quantization noise)
Write a program that estimates, using the `welch` function designed in exercise 9.1, the PSD of the quantization noise for several values of the quantization step. Remember that, using the notations of Chapter 7, page 271, under the hypothesis that the quantization is uniform and that the quantization noise is white, the quantization noise's PSD is equal to $q^2/12$ on the signal's entire band. Compare the theoretical and estimated values of the quantization noise's PSDs using a speech signal.

HINT: type:

```
%===== DSPQ.M
load phrase; Ac=max(y);
Lfft=1024; Fe=8000; fq=Fe*(0:Lfft-1)/Lfft;
for M=4:7           %===== Number of bits
   q=2*Ac/2^M;      %===== Quantization step
   yQ=round(y/q)*q; eQ=y-yQ;
   [sf gamma]=welch(eQ,Lfft,'rect',Lfft,0.95);
   plot(fq,10*log10(sf)); hold on
   sth=10*log10(q*q/12); plot([0 Fe],[sth sth],'-.')
end
hold off; set(gca,'xlim',[0 Fe/2]); grid
```

The results are shown in Figure 9.5. Notice that the theoretical values (horizontal full line) are in agreement with the estimates. ■

Exercise 9.3 (Spectral observation and oversampling)
Use the `welch` function designed in exercise 9.1 to estimate the spectra.

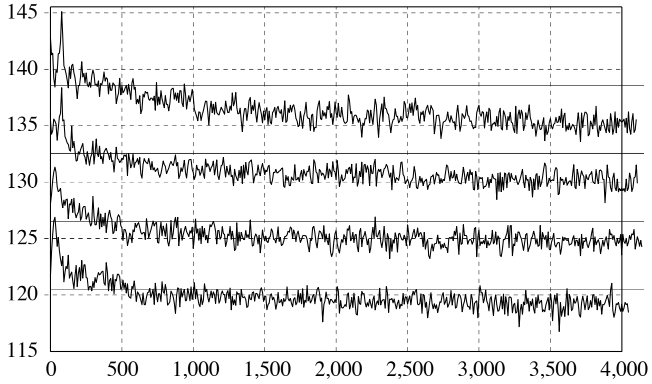


Figure 9.5 – *PSD of the quantization noise (in dB) as a function of the frequency (in Hz) for various values of the quantization step. The test signal is a speech signal sampled at* 8,000 *Hz. The dashed lines represent the theoretical value of the PSD under the hypothesis that the quantization noise is white*

1. Using filtering, create a signal $X(n)$ with the length 1024. A crude approximation could make this signal a model for audio-frequency signal. You can use the `rif` function (see page 599) with a small number of coefficients to generate this signal. Display the resulting signal's spectrum.

2. Perform the expansion operation corresponding to an oversampling by a factor of 8 (see page 151). Display the resulting signal's spectrum.

3. Perform the oversampling operation necessary to the filtering operation. You can use the `rif` function with a high number of coefficients. Display the resulting signal's spectrum.

9.2 Parametric estimation

9.2.1 AR estimation

The object of this section is to study methods for estimating the coefficients of an AR-model from the observed data $X(0)$, …,$X(N-1)$, and correlatively for estimating the dsp of this sample.

Least squares method

The first idea is to minimize, with respect to $\mathbf{a} = [a_1, \ldots, a_P]$, the criterion:

$$\textstyle\sum_{n=P}^{N-1}(X(n) - \sum_{j=1}^{P} a_j X(n-j))^2 \tag{9.12}$$

Canceling the derivatives w.r.t. $\{a_j\}$ gives:

$$\mathbf{a} = (\mathbf{D}^H\mathbf{D})^{-1}\mathbf{D}^H\mathbf{x}$$

where $\mathbf{x} = [X(P) \ \cdots \ X(N-1)]^T$ and (see equation 8.37):

$$\mathbf{D}^H = \begin{bmatrix} X^*(P-1) & X^*(P) & \cdots & X^*(N-2) \\ X^*(P-2) & X^*(P-1) & \cdots & X^*(N-3) \\ \vdots & \vdots & & \vdots \\ X^*(0) & X^*(1) & \cdots & X^*(N-P-1) \end{bmatrix} \tag{9.13}$$

The resulting matrix $\mathbf{D}^H\mathbf{D}$ is not Toëplitz due to end-effects. This might produce a polynomial $A(z)$ with zeroes outside the unit circle.

The Yule-Walker method

The Yule-Walker equations 8.58 provide us with a relation between the AR model's parameters and its covariance coefficients. This means they make it possible to estimate the parameters of an AR model by replacing the covariances with their estimates, provided for example by the equations 8.32.

The `[a,sigma2]=xtoa(X,P)` function given below estimates the sequence `a` of the P coefficients of an AR model as well as the power `sigma2` of the white noise input from a sample `X` and the model's assumed order P:

```
function [a,sigma2]=xtoa(x,P)
%%====================================================%
%% XTOA estimates the (P+1) parameters of an AR model %
%% SYNOPSIS: [a sigma2]=XTOA(x,P)                     %
%%     x      = Signal                               %
%%     P      = Order of the model                   %
%%     a      = [1 a_1 ..... a_P]                    %
%%     sigma2 = Variance of the input white noise    %
%%====================================================%
N=length(x); x=x(:); x=x-mean(x);
for kk=1:P+1
   rconj(kk)=x(kk:N)'*x(1:N-kk+1)/N;
end
Rc=toeplitz(rconj); vaux=Rc\eye(P+1,1);
a=vaux/vaux(1); sigma2=1/vaux(1);
return
```

In this program, the quantities `rconj(kk)` provide estimations for the covariances $R^*(k)$. We then know (see page 295) that the estimate `RC` of the covariance matrix, provided by the `toeplitz(rconj)` function, is a positive matrix.

To test the `xtoa` function, type the following program:

```
%===== TESTXTOA.M
trueCoef=[1 -1.3 0.8]; P=length(trueCoef)-1; sw=sqrt(1);
Lrun=100; listN=(500:500:4000); lgN=length(listN); perf=zeros(lgN,1);
for ii=1:lgN
   N=listN(ii);
   for ell=1:Lrun
      w=sw*randn(N,1); x=filter(1,trueCoef,w);
      [aest s2est]=xtoa(x,P);
      % Performance for the estimation of the coeff trueCoef(2)
      eQ=(aest(2)-trueCoef(2))*(aest(2)-trueCoef(2))';
      perf(ii)=perf(ii)+eQ;
   end
end
perf=perf/Lrun;
plot(listN,perf); hold; plot(listN,perf,'ro'); hold; grid
```

In this program, we consider the AR-2 process defined by $X(n) - 1.3X(n-1) + 0.8X(n-2) = W(n)$ where $W(n)$ is a white noise with a variance of 1. $L = 100$ runs are performed identically, and we will take the mean over L runs of the square deviation between the estimated value of a parameter and its true value as an indication of the estimation error. The experiment is repeated for several values N of the sample's length. Notice that the obtained graph shows that the deviation decreases when N increases. It can be shown that this deviation has the same asymptotic behavior as $1/\sqrt{N}$. The program can be modified to evaluate the estimation deviations on another of the model's parameters, or to change the model's order.

The Yule-Walker algorithm solves the least squares minimization by padding data with P zeroes on each side of the sequence (see equations 8.36 and 8.37), which is equivalent to add false data to the observed sequence. The main advantages are the following: the associated estimates define stationary processes and the estimates may be easily computed using the Levinson recursion.

Example 9.3 (Sunspot periodicity)

The sun's magnetic field, and its interactions with the movements of plasma, cause small, temporarily active regions called *sunspots* to appear on the surface. The intriguing part is how their number follows a cycle. The number of these sunspots has been recorded every month for over two centuries. The resulting values are available for research, and can be downloaded off the internet. The graph in Figure 9.6 shows the monthly data gathered from January 1750 to December 1999. As you can see, there are 23 lobes of roughly equal widths, over a period of 250 years. A tendency over a longer period is also visible, but is more difficult to analyze.

In 1898, Schuster defined the periodogram as a method to discover the frequencies of the "hidden harmonics" in a signal and was the first to use the periodogram to analyzing sunspot activity [85]. Later in 1927, the sunspot data have first been studied by Yule [105] with an AR model (see example 8.9).

Write a program to analyzing sunspot activity using periodogram and AR-2 estimation. Notice that it is necessary to denoise the signal before processing it. Use the function `rif.m` as a lowpass filter.

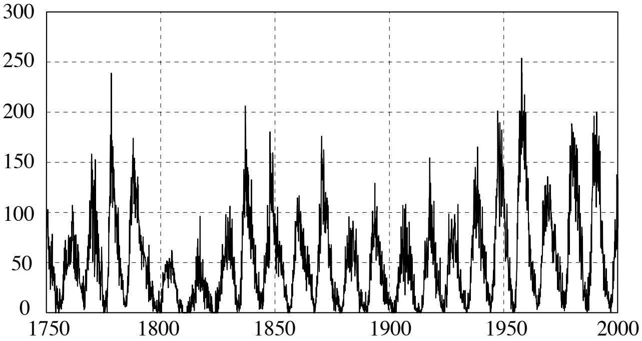


Figure 9.6 – *Number of sunspots recorded monthly during the period from January* 1750 *to December* 1999*, plotted against time*

HINT: type the program (Figure 9.7):

```
%===== ANALSUNSPOTS.M
clear; load tachsol; y=tachsol; N=length(y);
Fs=12; tps=(0:N-1)/Fs; % 12 samples per a year
[AA yc]=tendoff(y); Lyc=length(yc); % Detrend
h=rif(30,1/40); yc=filter(h,1,yc);
%===== Periodogram
Lfft=4096; fq=Fs*(0:Lfft-1)/Lfft;
perioy=abs(fft(yc,Lfft)).^2/N; [periomax indperiomax]=max(perioy);
f0=Fs*(indperiomax-1)/Lfft; mperiod=1/f0
%===== Covariance
P=2; rr=zeros(P+1,1);
for kk=1:P+1, rr(kk)=yc(kk:N)'*yc(1:N-kk+1)/N; end
Rc=toeplitz(rr(1:P+1)); vaux=Rc\eye(P+1,1);
%===== AR model
aAR=vaux/vaux(1); sigma2=1/vaux(1);
SAR=sigma2 ./ (abs(fft(aAR,Lfft)) .^2);
[SARmax indSARmax]=max(SAR);
f0_AR=Fs*(indSARmax-1)/Lfft; mperiodAR=1/f0_AR
%===== Figures
subplot(221); plot(yc(2:Lyc),y(1:N-1),'x');grid
subplot(222); plot3(yc(3:Lyc),yc(2:Lyc-1),y(1:N-2),'x'); grid
subplot(212); plot(fq,perioy,'g'); set(gca,'xlim',[0 Fs/30]); grid
hold on; plot(f0,periomax,'or');
```

```
plot(fq,SAR,'r'); set(gca,'xlim',[0 Fs/30]); hold off
```

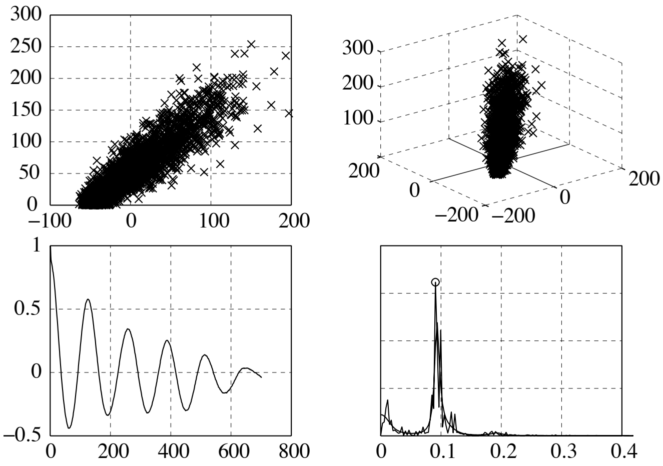


Figure 9.7 – *Sunspots: DTFT calculated over* 4,096 *frequency points*

Note that an estimation of the affine trend with function **tendoff** leads to $A(2) \approx 0$, as we can expect from to the slope from Figure 9.6, which is almost equal to zero. However, the mean is large since the signal's values are positive.

The processing, for the period, leads to a value slightly above 11 years. The previous program can be modified so as to estimate the period over subintervals of the set of data. This is a method that can be used for studying possible fluctuations. ■

The Burg method

J.P. Burg had the idea [16] [38] of directly estimating the reflection coefficients (see paragraph 8.5.3) from the data, and to do this *without first using the covariance calculation.* Once the reflection coefficients are calculated, the recursive equation 8.72 is used to determine the model's parameters.

To obtain the reflection coefficient estimates, J.P. Burg started off with the expression (see 8.60) of the error prediction sequence at a step $m \geq 0$:

$$\begin{aligned} \varepsilon_m^F(n) &= X(n) + a_m(1)X(n-1) + \cdots + a_m(m)X(n-m) \\ &= \mathbf{x}^T(n)\mathbf{a}_m^F \end{aligned}$$

for n from $m+1$ to N. This error is said to be *forward.* Also, the error said to be *backward* (see 11.34) is defined by:

$$\begin{aligned} \varepsilon_m^B(n) &= X^*(n-m) + a_m(1)X^*(n-m+1) + \cdots + a_m(m)X^*(n) \\ &= \mathbf{x}^H(n)\mathbf{a}_m^B \end{aligned}$$

Notice that it is the conjugated values of $X(n)$ that are involved in the definition of $\varepsilon_m^B(n)$. If we introduce the k_m starting at equation 8.72, we can write the m-th step forward error as a function of the $(m-1)$-th step errors:

$$\begin{aligned} \varepsilon_m^F(n) &= \mathbf{x}^T(n)\left(\begin{bmatrix} \mathbf{a}_{m-1}^F \\ 0 \end{bmatrix} + k_m \begin{bmatrix} 0 \\ \mathbf{a}_{m-1}^{B*} \end{bmatrix}\right) \\ &= \varepsilon_{m-1}^F(n) + k_m \varepsilon_{m-1}^{B*}(n-1) \end{aligned} \tag{9.14}$$

Likewise, we have for the backward error:

$$\begin{aligned} \varepsilon_m^B(n) &= \mathbf{x}^H(n)\left(\begin{bmatrix} 0 \\ \mathbf{a}_{m-1}^B \end{bmatrix} + k_m \begin{bmatrix} \mathbf{a}_{m-1}^{F*} \\ 0 \end{bmatrix}\right) \\ &= \varepsilon_{m-1}^B(n-1) + k_m \varepsilon_{m-1}^{F*}(n) \end{aligned} \tag{9.15}$$

These two relations will be proven in section 11.3 as part of our discussion on linear prediction.

In order to find an estimator of k_m, Burg suggests minimizing the sum of the errors $\mathbb{E}\{|\varepsilon_m^F(n)|^2 + |\varepsilon_m^B(n)|^2\}$ with respect to k_m. The two previous equations lead us to a second degree equation in k_m. If we set its derivative to zero, we get, for $m \geq 1$:

$$k_m = -\frac{2\mathbb{E}\{\varepsilon_{m-1}^B(n-1)\varepsilon_{m-1}^F(n)\}}{\mathbb{E}\{|\varepsilon_{m-1}^F(n)|^2 + |\varepsilon_{m-1}^B(n-1)|^2\}}$$

In the actual algorithm, an estimate of k_m is obtained with "temporal means". Thus, for $m \geq 1$, the numerator is estimated by:

$$\mathbb{E}\{\varepsilon_{m-1}^B(n-1)\varepsilon_{m-1}^F(n)\} \simeq \frac{1}{N-m+1}\sum_{n=m}^{N} \varepsilon_{m-1}^B(n-1)\varepsilon_{m-1}^F(n)$$

and the denominator by:

$$\begin{aligned} &\mathbb{E}\{|\varepsilon_{m-1}^F(n)|^2\} + \mathbb{E}\{|\varepsilon_{m-1}^B(n-1)|^2\} \\ &\qquad \approx \frac{1}{N-m+1}\left(\sum_{n=m}^{N} |\varepsilon_{m-1}^F(n)|^2 + \sum_{n=m}^{N} |\varepsilon_{m-1}^B(n-1)|^2\right) \end{aligned}$$

This is the Burg algorithm:

```
Initialization:
    $\varepsilon_0^F(n) = \varepsilon_0^B(n) = X(n)$ for $n \in \{1, \ldots, N\}$
    $a_0(0) = 1$ and $v_0 = \frac{1}{N}\sum_{n=1}^N |X(n)|^2$
For $m = 1, \ldots, K$, repeat:
    1. $k_m = \dfrac{-2\sum_{n=m}^N \varepsilon_{m-1}^B(n-1)\varepsilon_{m-1}^F(n)}{\sum_{n=m}^N |\varepsilon_{m-1}^B(n-1)|^2 + \sum_{n=m}^N |\varepsilon_{m-1}^F(n)|^2}$
    2. $a_m(0) = 1$ and $a_m(m) = k_m$
    3. For $j \in \{1, \cdots, m-1\} : a_m(j) = a_{m-1}(j) + k_m a_{m-1}^*(m-j)$
    4. $v_m = v_{m-1}(1 - |k_m|^2)$
    5. $\varepsilon_m^F(n) = \varepsilon_{m-1}^F(n) + k_m \varepsilon_{m-1}^{B*}(n-1)$ for $n = m+1 : N$
       $\varepsilon_m^B(n) = \varepsilon_{m-1}^B(n-1) + k_m \varepsilon_{m-1}^{F*}(n)$ for $n = m+1 : N$
```

One of the properties of the Burg algorithm is that it ensures that the reflection coefficient estimates have a modulus < 1, and therefore guarantees the stability of the *lattice* filter which will be described in Chapter 11. This helps to avoid introducing false data the way the correlation method does.

Exercise 9.4 (The Burg estimation of the AR parameters)

1. Write a function that implements the Burg algorithm.

2. Write a program that compares the results of the Burg algorithm with those obtained with the `xtoa` function from exercise 9.2.1.

The modified covariance method

If we stack the values of the forward error's expression for n from $p+1$ to $N-p$, we get:

$$\begin{aligned}\begin{bmatrix} X(p+1) \\ \vdots \\ X(N-p) \end{bmatrix} &= \begin{bmatrix} X(p) & \ldots & X(1) \\ \vdots & \ddots & \vdots \\ X(N-p-1) & \ldots & X(N-2p) \end{bmatrix} \begin{bmatrix} -a_1 \\ \vdots \\ -a_p \end{bmatrix} + \begin{bmatrix} \varepsilon_F(p+1) \\ \vdots \\ \varepsilon_F(N-p) \end{bmatrix} \\ &= -\mathbf{D}\mathbf{a} + \varepsilon_F \end{aligned} \tag{9.16}$$

One approach consists of choosing $\mathbf{a}^T = \begin{bmatrix} a_1 & \ldots & a_p \end{bmatrix}^T$ so as to minimize the norm of this error ε_F. We will see in Chapter 11 that the solution is given by formula 11.10 (page 396), which can be written:

$$\mathbf{a} = -(\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T\mathbf{X} \tag{9.17}$$

where $\mathbf{X} = \begin{bmatrix} X(p+1) & \dots & X(N-p) \end{bmatrix}^T$. This expression has to be compared with the Yule-Walker equations 8.58 that are constructed from the data matrices 8.36.

Starting off with the backward error's expression, we have:

$$\begin{aligned}\begin{bmatrix} X(p+1) \\ \vdots \\ X(N-p) \end{bmatrix} &= \begin{bmatrix} X(p+2) & \dots & X(2p+1) \\ \vdots & \ddots & \vdots \\ X(N-p+1) & \dots & X(N) \end{bmatrix} \begin{bmatrix} -a_1 \\ \vdots \\ -a_p \end{bmatrix} + \begin{bmatrix} \varepsilon_B(2p+1) \\ \vdots \\ \varepsilon_B(N) \end{bmatrix} \\ &= -\mathbf{B}\mathbf{a} + \varepsilon_B \end{aligned} \tag{9.18}$$

The *modified covariance* algorithm, or *Forward-Backward* algorithm, consists of minimizing the sum of the forward and backward errors, the way the Burg method did. If Δ denotes the half-sum of the forward and backward errors, we have:

$$\mathbf{X} = -\frac{1}{2}(\mathbf{D}+\mathbf{B})\mathbf{a} + \frac{1}{2}(\varepsilon_F + \varepsilon_B) = -\mathbf{H}\mathbf{a} + \Delta$$

If we minimize, we get:

$$\mathbf{a} = -(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{X} \tag{9.19}$$

The essential difference between this algorithm and the one suggested by Burg is that no stability constraint is imposed. However, it is more efficient when estimating sines, but it does not have the drawback of the correlation method's poor behavior for small samples.

Exercise 9.5 (AR-1 estimation and confidence intervals)
Let $\hat{a}$ and $\hat{\sigma}^2$ be the respective estimators of the parameters a and σ^2 of the AR-1 model. Because the estimators are functions of the random variables $X(1)$, ..., $X(N)$, they are themselves random variables. However, even in cases where $X(1)$, ..., $X(N)$ are assumed to be Gaussian, their probability distribution does not have a simple expression, but when N tends to infinity, one version of the central limit theorem states that the random variables $(\hat{a}, \hat{\sigma}^2)$ are jointly Gaussian, independent, with the means a and σ^2 respectively, and with the variances $(1-a^2)/N$ et $2\sigma^4/N$ respectively. The general result for an AR-P is given in [21].

1. For large values of N, determine the 98% confidence ellipse of the pair $(\hat{a}, \hat{\sigma}^2)$.

2. Write a program that performs $L = 500$ consecutive runs of a length $N = 1{,}000$ sample of an AR-1 process defined by $X(n) + aX(n-1) = W(n)$ with $a = -0.7$ and $\sigma^2 = 1$, then estimates a and σ^2, and finally displays the result in the 98% confidence ellipse. Check that roughly 2% of 500, that is 10 points, are outside the ellipse (see exercise 7.1).

9.2.2 Estimating the spectrum of an AR process

We saw in paragraph 8.5.2 that an AR process is defined as the only WSS solution to the recursive equation:

$$X(n) + a_1 X(n-1) + \cdots + a_P X(n-P) = W(n)$$

where $W(n)$ is a white, centered, WSS random process with the variance σ^2 and where the polynomial $A(z) = 1 + a_1 z^{-1} + \cdots + a_P z^{-P} \neq 0$ for $|z| \geq 1$. In this case, $X(n)$ has a causal expression as a function of $W(n)$.

Relation 8.58 between the model's parameters and the covariances provides, as we have explained, a simple way of estimating the parameters $a_1, \ldots, a_P$ and σ^2 by substituting the theoretical autocovariance sequence with the sequence of the covariance estimates. The spectrum can then be estimated using the formula:

$$\widehat{S}(f) = \frac{\widehat{\sigma}^2}{|1 + \widehat{a}_1 e^{-2j\pi f} + \cdots + \widehat{a}_P e^{-2j\pi P f}|^2}$$

This formula is obtained by replacing the model's parameters with the estimated parameters in expression 8.57 (page 306) of the spectrum obtained with the filtering formula 8.39 and by considering an AR process as the output of a filter with the complex gain $H(f) = 1/A(e^{2j\pi f})$ and white noise with the PSD σ^2 as its input. Remember that solving equation 8.58 leads to sequence of coefficients $a_1, \ldots, a_P$ such that the polynomial $A(z) \neq 0$ for $|z| \geq 1$.

This spectral estimation method is sometimes called the *high-resolution method*. Notice that it is not affected, unlike the periodogram method, by the $2/N$ limitation related to the time truncation. Once the parameters have been measured, the spectrum is known with an "infinite" resolution. This may seem surprising, but is simply due to the fact that *a priori* information is added when we say that the signal is a P order AR process.

An important obstacle to the AR identification however is the noticeable loss of resolution in the case of noised observations. The signal is then of the type $Y(n) = X(n) + B(n)$ where $X(n)$ is an AR process and $B(n)$ is a noise. It can be shown that for small signal-to-noise ratios, the periodogram's resolution is improved. [54] gives the following order of magnitude: if the signal-to-noise ratio, expressed in decibels, is less than $(32 \log_{10} N - 24)$, the periodogram's resolution is better than that of the AR method. For example, for $N = 100$, the formula indicates that if SNR < 40 dB, it is wiser to use the periodogram.

9.2.3 The Durbin method of MA estimation

Consider a process MA-Q defined by:

$$X(n) = W(n) + b_1 W(n-1) + \cdots + b_Q W(n-Q)$$

where $W(n)$ refers to a white, WSS, centered random process with the variance σ^2, $(b_1, \ldots, b_Q)$ is a sequence of Q coefficients. $X(n)$ is observed for $n = 1, \ldots, N$, and we wish to estimate, using the coefficients, the parameters of this model. Because the process is centered, the covariance estimates are denoted with:

$$\hat{R}(k) = \frac{1}{N} \sum_{j=1}^{N-k} X(j+k)X(j)$$

We know (see page 303 on MA-Q process) that the equation system, aside from not being linear, has more than one solution. The solution we are going to give leads to *minimum phase solution*, that is the one for which all of the roots of the polynomial $B(z) = 1 + b_1 z^{-1} + \cdots + b_Q z^{-Q}$ have a modulus < 1.

We wish here to indirectly estimate the parameters of the MA process by using the estimation for an AR process. Theoretically, as the inverse of $B(z)$, the polynomial of the exact AR process we are looking for should have an infinite length. In practice, the polynomial is chosen such that $A(z) = 1 + a_1 z^{-1} + \cdots + a_P z^{-P}$ with P high enough. The coefficient can then be estimated using the Levinson algorithm.

Once the sequence $\{1,\ a_1,\ a_2,\ \ldots,\ a_P\}$ is estimated, we express the fact that $A(z)B(z) \approx 1$, in other words the convolution of the sequence $\{1,\ a_1,\ a_2,\ \ldots,\ a_P\}$ with the sequence we are trying to determine $\{1,\ b_1,\ b_2,\ \ldots,\ b_Q\}$ is approximately $\delta(n)$. For example, we can minimize the ℓ^2 norm of the discrepancy:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ a_1 & 1 & & \vdots \\ \vdots & \ddots & & 1 \\ \vdots & & & a_1 \\ a_P & & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & & a_P \end{bmatrix} \begin{bmatrix} 1 \\ b_1 \\ \vdots \\ b_Q \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

If we call $\mathbf{A}$ the first matrix of this expression, then multiply on the left by $\mathbf{A}^H$, we get the expression $\mathbf{R}_A \begin{bmatrix} 1 & b_1 & \ldots & b_Q \end{bmatrix}^T - \begin{bmatrix} 1 & 0 & \ldots & 0 \end{bmatrix}^T$ where we have defined $\mathbf{R}_A = \mathbf{A}^H \mathbf{A}$. Notice that $\mathbf{R}_A$ is a positive Toeplitz matrix. This leads us to an equation similar to the Yule-Walker equation 8.58, which can therefore be solved with the Levinson algorithm presented in paragraph 8.5.3. This method was suggested by J. Durbin.

The choice of P essentially depends on where the zeros of the MA process are located: the closer they are to the unit circle, the higher P has to be. Finally, we wish to mention the fact that solving the Yule-Walker equations leads to a polynomial the zeros of which are inside the unit circle. Therefore, this method leads, for the MA process, to the minimum phase solution.

The following function estimates the parameters of an MA-Q using the Durbin method:

```
function [b,sigma2]=durbin(x,Q,P)
%%=====================================================%
%% DURBIN estimating the (Q+1) parameters of a MA process %
%% SYNOPSIS: [b,sigma2]=DURBIN(x,q)                    %
%%     x      = signal                                 %
%%     Q      = model order                            %
%%     b      = [1 b_1 ..... b_q]                      %
%%     sigma2 = power of the white noise input         %
%%=====================================================%
N=length(x); x=x(:); x=x-ones(1,N)*x/N;
if nargin<3, P=8*Q; end
for kk=1:P+1, rx(kk)=x(kk:N)'*x(1:N-kk+1)/N; end
Rx=toeplitz(rx); Phix=Rx\[1;zeros(P,1)]; sigma2=1/Phix(1);
for kk=1:Q+1,ra(kk)=Phix(kk:P+1)'*Phix(1:P-kk+2); end
Ra=toeplitz(ra); Phia=Ra\[1;zeros(Q,1)]; b=Phia/Phia(1);
return
```

The following program tests the result:

```
%===== TESTDURBIN.M
sigma2=8; w=sqrt(sigma2)*randn(10000,1);
hh=[1;0.3];    % minimum phase case
% hh=[.3;1];   % non minimum phase case
Q=length(hh)-1; x=filter(hh,1,w);
[b,s2]=durbin(x,Q); [hh b], [sigma2 s2]
```

You can change `hh=[1;0.3]` to `hh=[0.3;1]`, thus moving the zero from inside to outside the unit circle.

Example 9.4 (Estimating the PSD of an MA-1**)**
Using MATLAB®, write a program:

- that generates N values of an MA-1, WSS random process;
- that calculates the theoretical spectrum;
- that estimates the spectrum with the use of the `welch` function;
- that estimates the spectrum from an estimation of $R_{XX}(0), \ldots, R_{XX}(K)$ for $K = 2$ and $K = 4$ (`covtodsp` function);

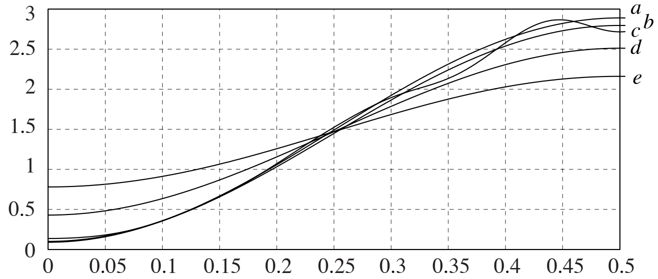


Figure 9.8 – *Estimating an MA-1 process's PSD*

– that estimates the spectrum using the Durbin method for $P = 15$.

Hint: type (see Figure 9.8):

```
%===== DSPMA1.M
N=3000; b1=[1;-0.7]; sigmaw2=1;
w=sigmaw2*randn(N,1); x=filter(b1,1,w);
Lfft=1024; b1s=fft(b1,Lfft);
fq=(0:Lfft-1)/Lfft; Sth=sigmaw2*abs(b1s) .^2;
%===== Welch method
Swelch=welch(x,16,'h',Lfft,.95);
%===== K=2 covariances
Scov2f=covtodsp(x,2,'b',Lfft).';
%===== K=4 covariances
Scov4f=covtodsp(x,4,'b',Lfft).';
%===== Durbin method
[b1ch, sigma2ch]=durbin(x,1,15);
Sdurb=sigma2ch*abs(fft(b1ch,Lfft)).^2;
plot(fq, [Sth Swelch Scov2f Scov4f Sdurb])
set(gca,'xlim',[0 0.5]); grid
```

Comment on Figure 9.8:

– (a) theoretical PSD;

– (b) PSD estimated with the Durbin method for an AR-15;

– (c) PSD estimated with the Welch method for a Hamming window with a length of 16;

– (d) PSD estimated as the DTFT of the sequence of 4 covariance coefficients estimated with the Bartlett window;

– (e) PSD estimated as the DTFT of the sequence of 2 covariance coefficients estimated with the Bartlett window.

The methods that directly use the covariance estimates (situations d and e) often leads to results that are not as good as those obtained with the Durbin method. ■

Chapter 10

Discrete Spectra Estimation

As we have seen for both the deterministic and the random cases, a signal composed of a sum of sines shows "peaks" in its spectrum. The object of this chapter is to study methods for estimating their frequencies and their amplitudes when the signal is corrupted by noise.

In this chapter, random processes will be denoted by lowercase letters so as to reserve capital letters for Fourier transforms.

10.1 Estimating the amplitudes and the frequencies

10.1.1 The case of a single complex exponential

Consider an observation $x(n) = s(n;\theta) + b(n)$ where $s(n;\theta) = \alpha_1 e^{2j\pi f_1 n}$ is a complex harmonic signal, where $b(n)$ is a white, centered, WSS, complex random signal, and where θ refers to the parameters (α_1, f_1). We are going to try to estimate the complex parameter α_1 and the parameter f_1, which belongs to $(0,1)$, based on a sequence of N noised observations.

In practice, the noise $b(n)$ is used to take into account the measurement errors, but also the possibility that we are not quite sure of the model used for the signal $s(n;\theta)$. This occurs when we have *a priori* information at our disposal on the wanted signal, for example with an active radar, where $s(n)$ represents the signal emitted then sent back by the target. It is also the case with speech when some of the noises originating from the vocal cords are described as a sum of sines.

The *least squares method*, the general presentation of which is given in Chapter 11, consists of calculating the values of α_1 and f_1 that minimize the square deviation between the observed values, that is $x(n)$, and the expected values, that is $s(n;\theta)$. When the noise is assumed to be Gaussian, the obtained values are those that maximize the probability density. In that case, the method is called the *maximum likelihood method.*

By stacking a sequence of N successive values of the model for the signal $s(n;\theta) = \alpha_1 e^{2j\pi n f_1}$, for n from 0 to $N-1$, we get the vector expression:

$$\mathbf{s}(\theta) = \alpha_1 \mathbf{e}(f_1)$$

$$\text{with } \begin{cases} \mathbf{s}(\theta) = [s(0) \quad \ldots \quad s(N-1)]^T \\ \mathbf{e}(f_1) = \begin{bmatrix} 1 & e^{2j\pi f_1} & \ldots & e^{2j\pi f_1(N-1)} \end{bmatrix}^T \end{cases}$$

Notice that the expression $\mathbf{s}(\theta) = \alpha_1 \mathbf{e}(f_1)$ is linear with respect to α_1 whereas it is not with respect to f_1.

The square deviation between the observation and the model is given by:

$$\begin{aligned} J(\alpha_1, f_1) &= \sum_{n=0}^{N-1} |x(n) - s(n;\theta)|^2 = (\mathbf{x} - \alpha_1 \mathbf{e})^H (\mathbf{x} - \alpha_1 \mathbf{e}) \\ &= (\mathbf{x}^H - \alpha_1^* \mathbf{e}^H)(\mathbf{x} - \alpha_1 \mathbf{e}) \end{aligned}$$

where $\mathbf{x} = [x(0) \quad \ldots \quad x(N-1)]^T$ represents the sequence of N observations. The expression is similar to the one we encountered in example 8.6 page 291, on suppressing seasonal trends. The minimization of $J(\alpha_1, f_1)$ with respect to α_1 and f_1 is performed first by setting to zero the derivative with respect to α_1, then by replacing the result in $J(\alpha_1, f_1)$. We get:

$$\frac{\partial J}{\partial \alpha_1} = 2\mathbf{e}^H (\mathbf{x} - \alpha_1 \mathbf{e}) = 0 \Leftrightarrow \mathbf{e}^H \mathbf{x} = \alpha_1 \mathbf{e}^H \mathbf{e}$$

Noticing that $\mathbf{e}^H \mathbf{e} = N$ for any f_1 leads us to:

$$\alpha_1 = \frac{1}{N} \mathbf{e}^H \mathbf{x} = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-2j\pi f_1 n} \tag{10.1}$$

By replacing this expression of α_1 in the expression of J, we get a new expression dependent only on f_1:

$$J_1(f_1) = \mathbf{x}^H \mathbf{x} - \alpha_1 \mathbf{e}^H \mathbf{x} = \mathbf{x}^H \mathbf{x} - \frac{1}{N} \left| \sum_{n=0}^{N-1} x(n) e^{-2j\pi f_1 n} \right|^2$$

and that we have to minimize with respect to $f_1 \in (0,1)$. Because the first term $\mathbf{x}^H \mathbf{x}$ does not depend on f_1, the problem is equivalent to determining the value of $f_1 \in (0,1)$ that maximizes the expression:

$$K(f_1) = \frac{1}{N} \left| \sum_{n=0}^{N-1} x(n) e^{-2j\pi f_1 n} \right|^2 \tag{10.2}$$

There is no simple analytical solution to this problem. It will be denoted by:

$$\widehat{f}_1 = \arg \max_{f_1 \in (0,1)} \frac{1}{N} \left| \sum_{n=0}^{N-1} x(n) e^{-2j\pi f_1 n} \right|^2$$

However, an approximation of the solution can be obtained digitally by performing a tightened sampling of the interval $(0,1)$. Once this value is calculated, we get the numerical complex amplitude using expression 10.1:

$$\widehat{\alpha}_1 = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-2j\pi \widehat{f}_1 n}$$

The fact that we used expression 10.2 to estimate a frequency is not in the least surprising, because it contains the expression of the DTFT of the sequence $\{x(0), \ldots, x(N-1)\}$ or to be more precise, its square modulus, which is the expression of the periodogram (definition 9.5).

To estimate, in the least squares sense, the frequency of one complex exponential corrupted by white noise, all we have to do is calculate the observation periodogram and find the frequency for which it reaches its maximum.

10.1.2 Real harmonic mixtures

We now consider a real signal, sum of P sinusoidal components of the type:

$$\begin{aligned} s(n) &= \sum_{k=1}^{P} A_k \cos(2\pi f_k n + \phi_k) \\ &= \sum_{k=1}^{P} a_k \cos(2\pi f_k n) + \sum_{k=1}^{P} b_k \sin(2\pi f_k n) \end{aligned} \tag{10.3}$$

where A_k, f_k and ϕ_k represent the parameters we wish to estimate based on a sequence of N observations. The frequencies f_1, ..., f_P are all assumed to be different.

Notice that we go from the pair $(A_k, \phi_k) \in \mathbb{R}^+ \times (0, 2\pi)$ to the pair $(a_k, b_k) \in \mathbb{R} \times \mathbb{R}$ using the bijection:

$$\left\{ \begin{array}{l} a_k = A_k \cos(\phi_k) \\ b_k = -A_k \sin(\phi_k) \end{array} \right. \Longleftrightarrow \left\{ \begin{array}{l} A_k = \sqrt{a_k^2 + b_k^2} \\ \phi_k = -\arctan(b_k / a_k) \end{array} \right. \tag{10.4}$$

From now on, we will assume that P is known, and that there are more observations than there are parameters to estimate. Let $\mathbf{f} = (f_1, \ldots, f_P)$, $\mathbf{s} = [s(0) \quad \ldots \quad s(N-1)]^T$, and:

$$\mathbf{A}(\mathbf{f}) = \begin{bmatrix}\mathbf{C}(\mathbf{f}) & \mathbf{S}(\mathbf{f})\end{bmatrix} \tag{10.5}$$

where:

$$\mathbf{C}(\mathbf{f}) = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ \cos(2\pi k f_1) & \cdots & \cos(2\pi k f_P) \\ \vdots & & \vdots \\ \cos(2\pi(N-1)f_1) & \cdots & \cos(2\pi(N-1)f_P) \end{bmatrix}$$

$$\mathbf{S}(\mathbf{f}) = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ \sin(2\pi k f_1) & \cdots & \sin(2\pi k f_P) \\ \vdots & & \vdots \\ \sin(2\pi(N-1)f_1) & \cdots & \sin(2\pi(N-1)f_P) \end{bmatrix}$$

$\mathbf{A}(\mathbf{f})$ is a $(N \times 2P)$ matrix. If we use these notations, and stack the N equations of $s(n)$ for n from 0 to $N-1$, we get the expression:

$$\mathbf{s} = \mathbf{A}(\mathbf{f})\mathbf{d} \tag{10.6}$$

where the $Q = 2P$ sized vector

$$\mathbf{d} = [a_1 \quad \ldots \quad a_P \quad b_1 \quad \ldots \quad b_P]^T = [d_1 \quad \ldots \quad d_Q]$$

is the amplitude vector we wish to estimate. The expression of the square deviation is still given by:

$$\begin{aligned} J(\mathbf{d}, \mathbf{f}) &= \sum_{n=0}^{N-1} |x(n) - s(n)|^2 = (\mathbf{x} - \mathbf{A}(\mathbf{f})\mathbf{d})^T(\mathbf{x} - \mathbf{A}(\mathbf{f})\mathbf{d}) \\ &= (\mathbf{x}^T - \mathbf{d}^T\mathbf{A}(\mathbf{f})^T)(\mathbf{x} - \mathbf{A}(\mathbf{f})\mathbf{d}) \end{aligned} \tag{10.7}$$

where $\mathbf{x} = [x(0) \quad \ldots \quad x(N-1)]^T$. To solve the problem, we are going to proceed as we did previously by setting to zero the partial derivatives of J with respect to each of the components of $\mathbf{d}$. We have:

$$\frac{\partial J}{\partial d_j} = \mathbf{a}_j^T(\mathbf{x} - \mathbf{A}(\mathbf{f})\mathbf{d}) = 0 \qquad j \in \{1, \ldots, Q\}$$

where $\mathbf{a}_j$ represents the j-th column of $\mathbf{A}$. If we group together the equations in matrix form, we get:

$$\mathbf{A}(\mathbf{f})^T(\mathbf{x} - \mathbf{A}(\mathbf{f})\mathbf{d}) = \mathbf{0} \Leftrightarrow \mathbf{A}(\mathbf{f})^T\mathbf{A}(\mathbf{f})\mathbf{d} = \mathbf{A}(\mathbf{f})^T\mathbf{x}$$

Because the values of f_1, ..., f_P are assumed to be all different, the matrix $\mathbf{A}(\mathbf{f})^T\mathbf{A}(\mathbf{f})$ is invertible. This leads to the expression of $\mathbf{d}$, dependent on f_1, ..., f_P, that leads to the maximum:

$$\mathbf{d} = [\mathbf{A}(\mathbf{f})^T\mathbf{A}(\mathbf{f})]^{-1}\mathbf{A}(\mathbf{f})^T\mathbf{x} \tag{10.8}$$

If we then replace this value of $\mathbf{d}$ in J, the resulting expression is a function of f_1, ..., f_P that we have to maximize:

$$\begin{aligned} J_1(f_1, \ldots, f_P) &= (\mathbf{x}^T - \mathbf{d}^T\mathbf{A}(\mathbf{f})^T)(\mathbf{x} - \mathbf{A}(\mathbf{f})\mathbf{d}) \\ &= \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{A}(\mathbf{f})[\mathbf{A}(\mathbf{f})^T\mathbf{A}(\mathbf{f})]^{-1}\mathbf{A}(\mathbf{f})^T\mathbf{x} \end{aligned}$$

where we have used the fact that the matrix$[\mathbf{A}(\mathbf{f})^T\mathbf{A}(\mathbf{f})]^{-1}$ is identical to its transpose. Because the first term is not dependent on the frequencies $\mathbf{f}$ we are trying to determine, the minimization is equivalent to the maximization of:

$$K(f_1, \ldots, f_P) = \mathbf{x}^T\mathbf{A}(\mathbf{f})\left[\mathbf{A}(\mathbf{f})^T\mathbf{A}(\mathbf{f})\right]^{-1}\mathbf{A}(\mathbf{f})^T\mathbf{x} \tag{10.9}$$

The expression of K contains the frequencies f_1, ..., f_P, but is not linear with respect to these frequencies. Just as before, its maximization does not lead to a simple analytical formula. We will simply write:

$$\widehat{\mathbf{f}} = \arg \max_{\mathbf{f} \in (0,1) \times \cdots \times (0,1)} \mathbf{x}^T\mathbf{A}(\mathbf{f})\left[\mathbf{A}(\mathbf{f})^T\mathbf{A}(\mathbf{f})\right]^{-1}\mathbf{A}(\mathbf{f})^T\mathbf{x}$$

We still have the possibility of a numerical calculation, but the problem quickly becomes overwhelming, because we have to find the maximum of a function of P variables. Once $\mathbf{f}$ has been calculated, the amplitudes are found by replacing its value in 10.8. We get:

$$\widehat{\mathbf{d}} = \left[\mathbf{A}(\hat{\mathbf{f}})^T\mathbf{A}(\hat{\mathbf{f}})\right]^{-1}\mathbf{A}(\hat{\mathbf{f}})^T\mathbf{x} \tag{10.10}$$

10.1.3 Complex harmonic mixtures

The complex case is dealt with in exactly the same way as the real case. Consider:

$$s(n) = \sum_{k=1}^{P} \alpha_k \exp(2j\pi f_k n) \tag{10.11}$$

where the α_k are a sequence of P complex amplitudes and the f_k are a sequence of P frequencies assumed to be all different, and belonging to the interval $(0, 1)$.

The square deviation between the observation and the model has the expression:

$$J(\alpha_1,\ldots,\alpha_P,f_1,\ldots,f_P)=\sum_{n=0}^{N-1}\left|x(n)-\sum_{k=1}^{P}\alpha_k e^{2j\pi f_k n}\right|^2 \tag{10.12}$$

If we let $\mathbf{f}=(f_1,\ldots,f_P)$ and:

$$\mathbf{E}(\mathbf{f})=\begin{bmatrix} 1 & 1 & \cdots & 1 \\ e^{2j\pi f_1} & e^{2j\pi f_2} & \cdots & e^{2j\pi f_P} \\ \vdots & \vdots & & \vdots \\ e^{2j\pi(N-1)f_1} & e^{2j\pi(N-1)f_2} & \cdots & e^{2j\pi(N-1)f_P} \end{bmatrix} \tag{10.13}$$

J can be written:

$$\begin{aligned} J(\alpha_1,\ldots,\alpha_P,f_1,\ldots,f_P) &= |\mathbf{x}-\mathbf{E}(\mathbf{f})\mathbf{a}|^2=(\mathbf{x}-\mathbf{E}(\mathbf{f})\mathbf{a})^H(\mathbf{x}-\mathbf{E}(\mathbf{f})\mathbf{a}) \\ &= (\mathbf{x}^H-\mathbf{a}^H\mathbf{E}(\mathbf{f})^H)(\mathbf{x}-\mathbf{E}(\mathbf{f})\mathbf{a}) \end{aligned}$$

where the exponent H indicates a transpose-conjugation. First, we minimize with respect to $\mathbf{a}=[\alpha_1,\,\ldots,\,\alpha_P]^T$. If we set to zero the partial derivatives, we get:

$$\mathbf{E}(\mathbf{f})^H(\mathbf{x}-\mathbf{E}(\mathbf{f})\mathbf{a})=\mathbf{0}\Leftrightarrow \mathbf{E}(\mathbf{f})^H\mathbf{x}=\mathbf{E}(\mathbf{f})^H\mathbf{E}(\mathbf{f})\mathbf{a}$$

You can check for yourself that if the P frequencies are different, the matrix $\mathbf{E}(\mathbf{f})^H\mathbf{E}(\mathbf{f})$ is invertible. This means that:

$$\mathbf{a}=\left[\mathbf{E}(\mathbf{f})^H\mathbf{E}(\mathbf{f})\right]^{-1}\mathbf{E}(\mathbf{f})^H\mathbf{x} \tag{10.14}$$

and that the minimum's expression is:

$$J_1=\mathbf{x}^H\mathbf{x}-\mathbf{x}^H\mathbf{E}(\mathbf{f})\left[\mathbf{E}(\mathbf{f})^H\mathbf{E}(\mathbf{f})\right]^{-1}\mathbf{E}(\mathbf{f})^H\mathbf{x}$$

We still have to minimize this quantity with respect to the set of frequencies $\mathbf{f}$. Because the first term $\mathbf{x}^H\mathbf{x}$ does not depend on $\mathbf{f}$, this is equivalent to maximizing the second term:

$$K(f_1,\ldots,f_P)=\mathbf{x}^H\mathbf{E}(\mathbf{f})\left[\mathbf{E}(\mathbf{f})^H\mathbf{E}(\mathbf{f})\right]^{-1}\mathbf{E}(\mathbf{f})^H\mathbf{x} \tag{10.15}$$

which implies the difficulties we mentioned earlier, and that will be studied in detail in the following paragraph.

10.2 Periodograms and the resolution limit

When P is greater than 1, it becomes difficult to maximize the function $K(f_1, \ldots, f_P)$ with respect to the frequencies f_1, ..., f_P, whether in the real case with expression 10.9, or in the complex case with expression 10.15, because it is a function of several variables, and usually has several local maxima.

However, if the differences between the frequencies are greater than $2/N$, the method whereby the P maxima of the periodogram are determined is a quite efficient method. This makes the calculations much simpler since the multi-variable maximization problem is changed into a single-variable maximization problem. This is what we are going to see now by numerically studying the case where $P = 2$.

Presence of several maxima for k

Let us reconsider, for $P = 2$, the expression of the function $K(f_1, f_2)$ defined by 10.13. The matrix $\mathbf{E}$ has the expression:

$$\mathbf{E}^H(f_1, f_2) = \begin{bmatrix} 1 & e^{-2j\pi f_1} & \ldots & e^{-2j\pi(N-1)f_1} \\ 1 & e^{-2j\pi f_2} & \ldots & e^{-2j\pi(N-1)f_2} \end{bmatrix}$$

This means that:

$$\mathbf{E}^H\mathbf{E} = N \begin{bmatrix} 1 & p_N(f_2 - f_1) \\ p_N^*(f_2 - f_1) & 1 \end{bmatrix}$$

where:

$$p_N(f) = e^{j\pi f(N-1)} \frac{\sin(\pi N f)}{N \sin(\pi f)} \tag{10.16}$$

By replacing this result in expression 10.15, we get:

$$\begin{aligned} K(f_1, f_2) &= \frac{1}{N} \begin{bmatrix} X_N^*(f_1) & X_N^*(f_2) \end{bmatrix} \\ &\times \begin{bmatrix} 1 & p_N(f_2 - f_1) \\ p_N^*(f_2 - f_1) & 1 \end{bmatrix}^{-1} \begin{bmatrix} X_N(f_1) \\ X_N(f_2) \end{bmatrix} \end{aligned} \tag{10.17}$$

where:

$$X_N(f) = \sum_{n=0}^{N-1} x(n) e^{-2j\pi f n}$$

is simply the DTFT of the sequence $x(n)$. We are going to perform a numerical study by considering the signal $x(n) = s(n) + b(n)$ where:

$$s(n) = a_1^0 e^{2j\pi f_1^0 n} + a_2^0 e^{2j\pi f_2^0 n}$$

with $a_1^0 = 1.5$, $a_2^0 = 1$, $f_1^0 = 0.12$ and $f_2^0 = 0.61$ and where $b(n)$ is a white, centered, Gaussian noise. There are $N = 10$ sample values. The following program generates the signal $x(n)$, plots the surface $K(f_1, f_2)$ defined by 10.15 as well as the periodogram of $x(n)$ defined by:

$$I_N(f) = \frac{1}{N} |X_N(f)|^2 \tag{10.18}$$

Type:

```
%===== CMLE.M
clear; T=10; f01=.12; f02=.61; tps=[0:T-1]';
%===== Signal
s=1.5*exp(2*j*pi*f01*tps)+exp(2*j*pi*f02*tps);
SNR=15; sigma2= (s'*s/T)/(10 ^(SNR/10));
%===== Noised signal
xb=s+sqrt(sigma2)*randn(T,1);
Lf=70; f1=(0:Lf-1)/Lf; f2=f1;
mm=exp(2*j*pi*tps*f1);
%===== or: [X, Y]=meshgrid(f1,tps); mm=exp(2*j*pi*(X.*Y));
yy=zeros(Lf,Lf);
for k1=1:Lf
    for k2=1:k1-1
        E=[mm(:,k1) mm(:,k2)];
        yy(k1,k2)=abs(xb' * E * pinv(E)  * xb);
    end
end
subplot(121); mesh(f1,f2,yy); view([115 35])
subplot(122); plot(f1,abs(fft(xb,Lf))); grid
```

The results, obtained for a signal-to-noise ratio equal to 15 dB are shown in Figure 10.1. Because $K(f_1, f_2) = K^*(f_2, f_1)$, we restricted the representation to the half-plane delimited by the bisector of the first quadrant. The function shows a global maximum in M the coordinates of which, $\widehat{f}_1$ and $\widehat{f}_2$, are almost equal to the two real values. But this function also has local maxima such as m, making it difficult to find the global maximum using a numerical technique.

The graph at the bottom of Figure 10.1 shows that the obtained values $\widehat{f}_1$ and $\widehat{f}_2$ are almost equal to the x-coordinates of the two highest maxima of the periodogram of $x(n)$. As we are going to show, this property has to do with the fact that the frequency difference is such that $|f_1^0 - f_2^0|N = 4.5 \gg 1$. This is a fundamental result, because it justifies the use of the periodogram for estimating frequency sequences. Note that there is an essential difference *between the search for the global maximum of the multivariable function $K(f_1, f_2)$ and the search for the two highest maxima of the single-variable function $I_N(f)$.*

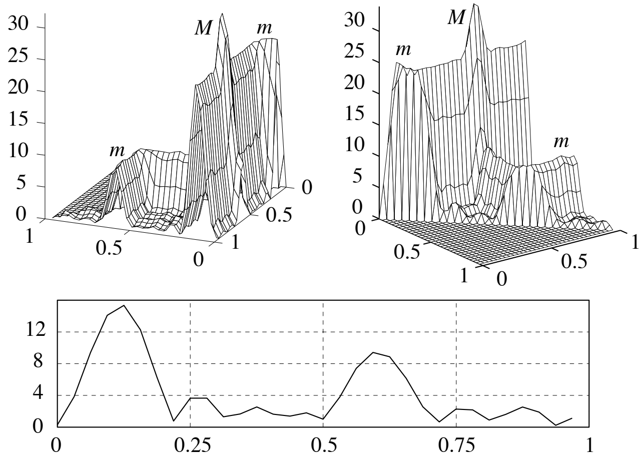


Figure 10.1 – *Frequency estimation: the figures correspond to the function $K(f_1, f_2)$ given by equation 10.17. The bottom figure shows the periodogram (equation 10.18) for $N = 10$ and a signal-to-noise ratio of* 15 *dB*

Resolution limit of Fourier

We are going to show that, when the frequencies contained in the signal $s(n)$ are such that:

$$\min_{\{i,j;i\neq j\}} |f_i^0 - f_j^0| \gg \frac{1}{N} \tag{10.19}$$

then the global maximum of $K(f_1, \ldots, f_P)$ is located at a point whose coordinates are almost equal to the periodogram's P highest maxima. First, let us write once more expression 10.15:

$$K(f_1, \cdots, f_P) = \mathbf{x}^H \mathbf{E} \left(\mathbf{E}^H \mathbf{E}\right)^{-1} \mathbf{E}^H \mathbf{x}$$

If condition 10.19 is met, then according to 10.13, the diagonal elements of $\mathbf{E}^H\mathbf{E}$ are equal to N and the non-diagonal elements have the expression:

$$p_N(f_k^0 - f_m^0) = e^{j\pi(N-1)(f_k^0 - f_m^0)} \frac{\sin(\pi N(f_k^0 - f_m^0))}{N\sin(\pi(f_k^0 - f_m^0))}$$

Therefore, they quickly tend to 0 when $(f_k^0 - f_m^0)N$ becomes large. Hence we can write, using the notation $\mathbf{E} = [\mathbf{e}_1, \cdots \mathbf{e}_P]$ in the form of column vectors:

$$\begin{aligned} K(f_1, \ldots, f_P) &\approx \frac{1}{N}\mathbf{x}^H \mathbf{E}\mathbf{E}^H \mathbf{x} = \frac{1}{N}\mathbf{x}^H [\mathbf{e}_1 \ldots \mathbf{e}_P] \begin{bmatrix} \mathbf{e}_1^H \\ \vdots \\ \mathbf{e}_P^H \end{bmatrix} \mathbf{x} \\ &= \frac{1}{N}\left(\mathbf{x}^H \mathbf{e}_1 \mathbf{e}_1^H \mathbf{x} + \cdots + \mathbf{x}^H \mathbf{e}_P \mathbf{e}_P^H \mathbf{x}\right) \end{aligned}$$

But according to 10.18, $\frac{1}{N}\mathbf{x}^H\mathbf{e}_j\mathbf{e}_j^H\mathbf{x} = I_N(f_j)$, and therefore:

$$K(f_1, \ldots, f_P) \approx I_N(f_1) + \cdots + I_N(f_P)$$

This function's maximum is obtained by separately maximizing $I_N(f)$ for each variable, hence the maximum of $K(f_1, \ldots, f_P)$ is obtained by using the periodogram as an univariate function. This is why the $2/N$ limit condition on the use of the periodogram is called the *fundamental resolution limit of Fourier.* Finally, the estimates, denoted by $\widehat{f}_k$, are used to simplify formula 10.14 and lead to the following estimates for the complex amplitudes:

$$\widehat{\alpha}_k = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-2\pi jn\widehat{f}_k} \tag{10.20}$$

We can also prove that, under condition 10.19 set by the resolution limit of Fourier, the result is similar in the case of real harmonic signals. Based on 10.5, we first show that:

$$\mathbf{A}(\mathbf{f})^T\mathbf{A}(\mathbf{f}) = \begin{bmatrix}\mathbf{C}(\mathbf{f})^T \\ \mathbf{S}(\mathbf{f})^T\end{bmatrix}\begin{bmatrix}\mathbf{C}(\mathbf{f}) & \mathbf{S}(\mathbf{f})\end{bmatrix} \simeq \frac{N}{2}\begin{bmatrix}\mathbf{I}_P & \mathbf{0}_P \\ \mathbf{0}_P & \mathbf{I}_P\end{bmatrix}$$

As a consequence, the frequencies are provided by the periodogram's maximum in the $(0, 1/2)$ band (because of the hermitian symmetry). As for the amplitudes, all we have to do is replace $\mathbf{A}(\mathbf{f})^T\mathbf{A}(\mathbf{f}) \approx (N/2)\mathbf{I}$ in 10.10, then use relations 10.4 and 10.6. We get:

$$\begin{bmatrix}\widehat{a}_k \\ \widehat{b}_k\end{bmatrix} = \frac{2}{N}\begin{bmatrix}\mathbf{C}^T \\ \mathbf{S}^T\end{bmatrix}\mathbf{x} = \frac{2}{N}\begin{bmatrix}\sum_{n=0}^{N-1} x(n)\cos(2\pi\widehat{f}_k n) \\ \sum_{n=0}^{N-1} x(n)\sin(2\pi\widehat{f}_k n)\end{bmatrix}$$

As a conclusion, the amplitudes a_k and $-b_k$ are twice the real and imaginary parts respectively of the complex quantities α_k given by expression 10.20.

Consider the case of a signal containing only one real sine with the frequency $f_1 = 0.1$, that is to say two complex exponentials with the frequencies f_1 and $-f_1$. If we apply the previous result, the periodogram is effective so long as the difference in frequency is such that $2f_1N \gg 1$. Let us assume that $N = 100$, meaning that $2f_1N = 20$. The following program implements the frequency estimation based on the periodogram:

```
%===== EST1SINREEL
clear; f1=0.1; Lfft=4*1024; N=100; A1=2; phi1=pi/3;
SNR=30; sigmab=A1*10^(-SNR/20);
xt=A1*cos(2*pi*f1*(0:N-1)+phi1)+sigmab*randn(1,N);
a1=A1*cos(phi1); b1=-A1*sin(phi1);
xf=fft(xt,Lfft)/N; [amp f1ind]=max(abs(xf(1:Lfft/2)));
f1est=(f1ind-1)/Lfft;
a1est=2*real(xf(f1ind)); b1est=-2*imag(xf(f1ind))
[f1 f1est], [a1 a1est], [b1 b1est], [A1 sqrt(a1est^2+b1est^2)]
```

Notice that the frequency value is given by the x-coordinate of the maximum of $I_N(f)$ on the interval $(0,1)$. The maximum can be found using the FFT-based computation of $I_N(f)$ for $f = k/L$ and $k \in \{0,\dots,L-1\}$. The frequency estimate's accuracy improves as the number L of FFT calculation points increases. This accuracy has direct consequences on the amplitudes of the sine and cosine components, and particularly on the measurement of the phase, as you can see from the two displayed values.

The performances are "enhanced" when N increases

We will show that, on average, the higher N is, the better the periodogram's components stand out in the noise. Consider once again the case where $P = 1$. In the presence of noise, the signal has the expression:

$$x(n) = a_1 e^{2j\pi f_1 n} + b(n)$$

where $n \in \{0, \dots, N-1\}$ and where $b(n)$ is assumed to be white with the variance σ^2. We are going to determine the periodogram's expression. Let:

$$p_N(f) = e^{j\pi(N-1)f} \frac{\sin(\pi N f)}{N \sin(\pi f)}$$

and:

$$B_N(f) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} b(n) e^{-2j\pi f n}$$

With these notations, the periodogram can be written:

$$\begin{aligned} I_N(f) &= \frac{1}{N} \left| \sum_{n=0}^{N-1} x(n) e^{-2j\pi n f} \right|^2 = \left| \sqrt{N} a_1 p_N(f - f_1) + B_N(f) \right|^2 \\ &= N a_1^2 p_N^2 (f - f_1) \\ &\qquad + 2\mathrm{Re}\{\sqrt{N} a_1 p_N(f - f_1) B_N^*(f)\} + |B_N(f)|^2 \qquad (10.21) \end{aligned}$$

Consider the expectation of $I_N(f)$. The first term is deterministic. The second term of 10.21 has an expectation equal to zero because $b(n)$ is centered. The expectation of the third term is expressed:

$$\mathbb{E}\{|B_N(f)|^2\} = \frac{1}{N} \sum_{n=0}^{N-1} \sum_{n'=0}^{N-1} \mathbb{E}\{b(n) b^*(n')\} e^{-2j\pi f n} e^{2j\pi f n'} = \sigma^2$$

where we have used the fact that $\mathbb{E}\{b(n)b^*(n')\} = \sigma^2 \mathbb{1}(n = n')$. Therefore:

$$\mathbb{E}\{I_N(f)\} = N a_1^2 p_N^2 (f - f_1) + \sigma^2$$

As a conclusion, $\mathbb{E}\{I_N(f)\}$ is comprised of two terms: the first one, related to the wanted signal, shows a maximum in f_1 that increases with N. The second one, related to the noise, is *independent* of N and equal to σ^2. When N increases, that part corresponding to the wanted signal tends to stand out in the noise around f_1. This can be checked by using the previous program. Theoretically, when N is multiplied by 2, there is a 3 dB gain on the emergence of the peak. This result can be generalized to the case of a signal containing P complex exponentials.

To sum up, if the frequency differences are much greater than $2/N$, determining the frequencies is equivalent to studying the periodogram, which shows:

– peaks around the real frequencies f_1^0, ..., f_P^0, the heights of which increase proportionally to N;

– and farther away from these frequencies, a basically "constant" level for the power of the noise.

Remember that using windows other than the rectangular windows makes it possible to reduce the height of the side lobes and therefore to help the low amplitude components stand out better, but at the cost of a worse frequency separation.

A program for the search of the P maxima

As we just saw, the P frequencies contained in a noised signal can be estimated by choosing the periodogram's P maxima. However, this is difficult to implement because the periodogram usually has local maxima that must not be taken into account. To solve this problem, we are going to use the fact that the periodogram of a sum of P sines, in the absence of noise, and under the Fourier condition (condition 10.19) of P lobes with the width Δf around the frequencies. The value of Δf essentially depends on the weighting window.

Example 10.1 (Finding the maxima)
Consider the length $N = 25$ sample of the signal $x(n) = s(n) + b(n)$ where $b(n)$ refers to a Gaussian, additive, white noise of unknown power σ^2. We know that $s(n)$ is the sum of 3 real sines with unknown frequencies and amplitudes and that the differences in frequency are much greater than the resolution limit of Fourier, which in this case is equal to $2/25 = 0.08$.

Write a program that estimates the three frequencies by calculating the periodogram over $L = 256$ frequency points. Remember that the choice of L is related to the frequency accuracy: with $L = 256$, for example, the calculation points on the spectrum are separated by $1/256 \approx 0.004$. The method mentioned previously consists of finding the first maximum then to eliminate the points in the spectrum around this maximum in a range of $\Delta f = \pm\alpha/N$. The choice of the value for α is based on the type of weighting window. For the rectangular window for example, the value of α is slightly higher than 1. The procedure is repeated until the P maxima are found.

Hint:

```
%===== PMAXSIN.M
clear; clg
P=3; A=[2 1.5 1]; F=[0.1;0.23;0.3]; N=25; Lfft=256;
deltaf=round(Lfft/N);
s=A*cos(2*pi*F*(0:N-1));
sigma2=0.5;x=s+sqrt(sigma2)*randn(1,N);
%===== Signal
subplot(221); plot(x); axis([0 N -5 5]); grid
text(16,4,'Temps')
%===== Spectrum
xf=abs(fft(x,Lfft)) .^2; xf=xf(1:Lfft/2)/max(xf);
subplot(222); plot((0:Lfft/2-1)/Lfft,xf);
set(gca,'xlim',[0 0.5],'ylim',[0 1]); grid
text(.25,.75,'Frequence')
subplot(212); grid
%===== Looking for the frequencies
for ii=1:P
    [mm im]=max(xf); fs=(im-1)/Lfft;
    u1=max(1,im-deltaf);u2=min(Lfft/2,im+deltaf);
    nb=u2-u1+1; xf(u1:u2)=zeros(1,nb);
    hold on; plot(fs,mm,'o'); hold off
    axis([0 0.5 0 1])
    text(fs+0.01,mm,sprintf('% .3g',fs))
end
```

Figure 10.2 shows the temporal form of the signal and its spectrum. At first, the periodicities are difficult to distinguish in the signal's representation as a function of time. The spectrum, on the other hand, clearly shows the location of the three frequencies. This calls for a comment: a more relevant study of the temporal signal consists of interpolating the signal (this is allowed because it is in agreement with the sampling theorem) so as to obtain a time discretization sufficient to estimate the periodicity.

The previous program also allows you to check the efficiency loss when the differences in frequency are too small. This is achieved by choosing values of Δf with a modulus close to $1/N$. ■

Example 10.2 (Analysis of a musical note)
Figure 10.3 shows the signal created by a piano note, a "C2" (264 Hz), sampled at the frequency 24 kHz. The signal's shape, as well as the physical phenomena involved when the chords vibrate, lead us to describing the signal as a sum of sines.

The spectrum in Figure 10.4, for a 600 samples portion of the signal, shows peaks with frequencies that are approximatively the multiples of a fundamental frequency corresponding to the note that is played. The *spectral envelope*, the

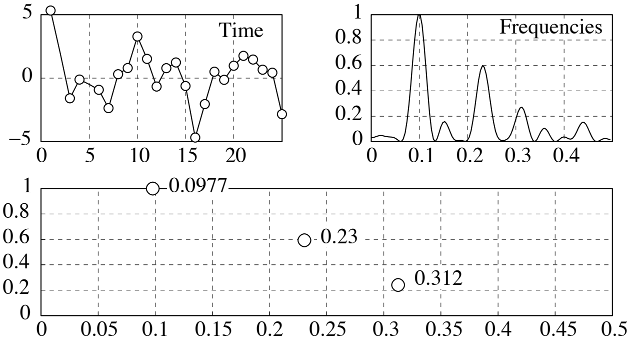


Figure 10.2 – *Top-left graph: the signal as a function of time. Top-right graph: the signal's spectrum. Bottom graph: location of the* 3 *maxima obtained with the program. The signal-to-noise ratio is equal to* 10 *dB and the real frequencies are equal to* 0.1, 0.23 *and* 0.3

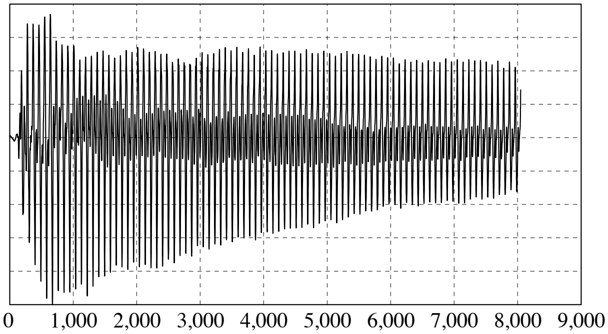


Figure 10.3 – *Note played by a piano*

virtual line that passes through the maxima of the peaks, is characteristic of the instrument's timbre.

We are first going to analyze the signal in such a way as to extract the main frequential components, then, based on these components, synthetize a signal. The point of this process is to have a small number of parameters (the duration, the frequency, the timbre) that can then be modified, to create a sound. After having recorded a sound created by a musical instrument such as a piano, a guitar, etc. write a program:

- that cuts up the signal in windows covering a duration of a few periods;
- that extracts the amplitudes and the frequencies of the P most important components. You can use the Hamming window, and with a method similar to the one used in the previous program, spread out a certain

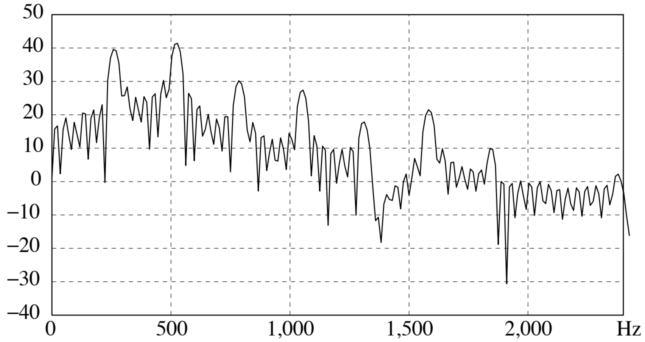


Figure 10.4 – *Spectrum of a* 600 *sample portion of the signal shown in Figure 10.3*

number of points located on either side of the maxima. The value of P can be chosen based on an *a priori* study or through an automatic process such as the one we will see on page 357;

– that creates a signal from the extracted amplitudes and frequencies. Discontinuities appear in the trajectory when the created portions are placed one after the other, and these discontinuities are distinctly audible. This problem can be solved by cutting up the signal in windows with an $\alpha\%$ overlap. When the signal is created, the calculated window is *multiplied by a triangular or trapezoidal window* then added with an overlap of $\alpha\%$ to the previous window (Figure 10.5).

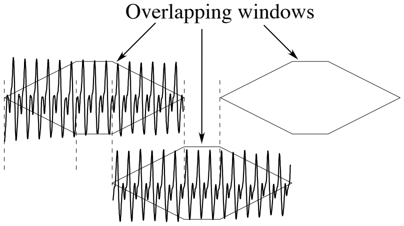


Figure 10.5 – *Reconstruction with overlapping blocks*

This "Overlap-Add" technique ensures a satisfactory continuity of the total trajectory.

HINT: the following program analyzes and synthetizes a piano note. The duration of a window was set to 350 samples, which corresponds for this note to a little over 4 periods. The window must be chosen long enough, but not too long to maintain a good stationarity. A long window is particularly badly suited for the attack and the release of a note.

```
%===== ANANOTE.M
clear
figure(1)
load piano; Fe=24000; N=length(piano);
%===== Splitting in blocks
lbloc=350; nbblocs=fix(N/lbloc);
pianoF=piano(1:nbblocs*lbloc);
xsyn=zeros(nbblocs*lbloc,1);
tpsbloc=(0:lbloc-1)/Fe;
%===== Windows
fenH=0.54-0.46*cos(2*pi*(0:lbloc-1)'/(lbloc-1));
fenH=fenH*lbloc/sum(fenH);      % Normalization
fenT=2*[(0:lbloc/2-1)'; (lbloc/2-1:-1:0)']/lbloc;
%===== Parameters of the spectral analysis
P=12; Lfft=4096; deltaf=2*round(Lfft/lbloc);
fq=Fe*(0:Lfft/2-1)/Lfft;
%===== Processing
for jj=0:2*nbblocs-2
    jj1=(lbloc/2)*jj+1; jj2=jj1+lbloc-1;
    x=pianoF(jj1:jj2) .* fenH; x=x-mean(x);
    fs=zeros(1,P); mm=zeros(P,1);
    %===== Spectrum
    xf=fft(x,Lfft); xf=xf(1:Lfft/2)/lbloc; xfvar=xf;
    xfvar(1:deltaf)=zeros(1,deltaf);
    %===== Analysis
    for ii=1:P
        [bid im]=max(abs(xfvar));
        fs(ii)=(im-1)/Lfft; mm(ii,1)=xfvar(im);
        u1=max(1,im-deltaf);u2=min(Lfft/2,im+deltaf);
        nb=u2-u1+1; xfvar(u1:u2)=zeros(1,nb);
    end
    %===== Synthesis
    xsyn_f=2*real(exp(2*j*pi*(0:lbloc-1)'*fs)*mm);
    %===== Overlap-add
    xsyn(jj1:jj2)=xsyn(jj1:jj2)+ xsyn_f .* fenT;
    subplot(211);
    plot(tpsbloc,pianoF(jj1:jj2),':',tpsbloc,xsyn_f); grid
    %===== Drawing the spectra
    subplot(212),plot(fq,20*log10(abs(xf)));
    set(gca,'ylim',[-70 0])
    hold on; plot(fs*Fe,20*log10(abs(mm)),'or'); hold off;
    grid; pause
end
ti=(0:nbblocs*lbloc-1);
%===== Displaying the reconstructed signal
figure(2); plot(ti,pianoF,'b',ti,xsyn,'r'); grid
```

The diagram at the bottom of Figure 10.6 shows the spectra of a signal portion and of the estimated frequencies and amplitudes. The graph above it shows the analyzed (dashed line) and synthetized signal (full line). ■

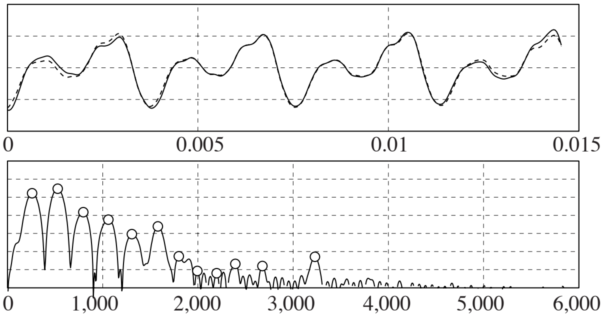


Figure 10.6 – *Frequency and amplitude estimates of the harmonic part for a window of* 350 *samples extracted from the signal shown in Figure 10.3. Top figure: the original signal (dashed line) and the synthetized signal (full line). Bottom figure: the periodogram. The dots ('o') indicate frequency and amplitude estimates*

We wish to determine sinusoidal frequencies, but in most problems, the number of sines is unknown. We can mention the case of the number of significant frequential components in a music signal (see example 10.2) or the case of the RADAR where P represents the number of targets that are being tracked. Unfortunately, estimating P is a difficult problem, and the reader can find more detailed information in the literature [54]. We will now introduce a heuristic method that has the advantage of being simple. It is based on the comment made on page 352 explaining that the periodogram's level is basically equal to the noise levels at the frequencies other than the sine frequencies. We are also going to estimate the value of σ^2 and use it to estimate the number of sines:

- Let us assume that the number of sines is less than a set value $P_{\max}$, given *a priori*. Its value depends on the practical information available concerning the system being studied.

- The periodogram is computed over L FFT points, and the values located in a Δ-wide interval around each of the $P_{\max}$ maxima. This makes it possible to eliminate the sinusoidal contributions. The choice of Δ depends on the type of window used and the number of FFT points. It can be adjusted depending on the situation. Usually the number of lobes is chosen to be an integer.

- Based on the remaining values of the periodogram, σ^2 is estimated.

- Finally, only the P maxima greater than a certain threshold are kept. The choice of this threshold can be made according to the *3-sigma rule*.

In terms of power, this leads the values of the periodogram smaller than $9\sigma^2$ to be considered as noise, and the rest as part of the signal.

```
%===== NBSIN.M
%===== Definition of the signal
A=[2 1.5 1]; F=[0.1;0.23;0.3]; N=100; Lfft=128;
deltap=3*round(Lfft/N);  % Value to adjust
                         % typically 3*Lfft/N
%===== Generation of samples
s=A*cos(2*pi*F*(0:N-1)); sigma2=0.4;
Pmax=6;                  %==== Number of sines
mm=zeros(1,Pmax); x=s+sqrt(sigma2)*randn(1,N);
xf=abs(fft(x,Lfft)) .^ 2 / N ;
xfplus=xf(2:Lfft/2);
for ii=1:Pmax
    [mm(ii) im]=max(xfplus);
    u1=max(1,im-deltap); u2=min(Lfft/2,im+deltap);
    nb=u2-u1+1;
    %===== Set to 0 close to the maxima
    xfplus(u1:u2)=zeros(1,nb);
end
nbz=length(find(xfplus==0));
%===== Mean of the values of the periodogram
sigma2est=sum(xfplus)/((Lfft/2)-nbz);
seuil=9*sigma2est;
P=length(find(mm>seuil));
disp(sprintf('%2i sines',P))
```

10.3 High resolution methods

Unlike the methods using the periodogram, even with windowing, the high resolution methods are such that the error tends to zero when SNR $\rightarrow \infty$.

10.3.1 Periodic signals and recursive equations

Property 10.1 *Let $s(n)$ be the harmonic signal such that:*

$$s(n) = \sum_{k=1}^{P} \alpha_k \exp(2j\pi f_k n) \tag{10.22}$$

where α_k is a sequence of P complex amplitudes and f_k is a sequence of P frequencies, all of them different from one another. Then there is a sequence $b_1, \dots, b_P$ such that:

1. *the signal $s(n)$ obeys the recursive equation:*

$$s(n) + b_1 s(n-1) + \cdots + b_P s(n-P) = 0 \tag{10.23}$$

2. and the equation:

$$B(z) = z^P + b_1 z^{P-1} + \cdots + b_P = 0 \tag{10.24}$$

has its P distinct roots on the unit circle.

Conversely, if $s(n)$ obeys 10.23, and if $B(z) = z^P + b_1 z^{P-1} + \cdots + b_P$ has its P distinct roots on the unit circle, then $s(n)$ is of the type 10.22 where the α_k are any P complex values.

The proof of this is in every way the same as the one given in Chapter 8, page 409 for random harmonic processes. It leads to $B(z) = \prod_{k=1}^{P}(z - z_k)$ where $z_k = e^{2j\pi f_k}$.

This is even more true when $s(n)$ is a real signal, sum of P *sines* of the type:

$$\begin{aligned} s(n) &= \sum_{k=1}^{P} a_k \cos(2\pi f_k n + \phi_k) \\ &= \sum_{k=1}^{P} \frac{a_k}{2}\left[e^{j\phi_k}\exp(2j\pi f_k n) + e^{-j\phi_k}\exp(-2j\pi f_k n)\right] \\ &= \sum_{k=1}^{2P} \alpha_k \zeta_k^n \end{aligned}$$

where the ζ_k are the $2P$ values of the type $e^{\pm 2j\pi f_k}$. This is because $s(n)$ is expressed as the sum of $2P$ complex exponentials the frequencies of which come in pairs of positive and negative values. Therefore, according to property 10.1, $s(n)$ obeys a recursive equation of the type $s(n)+b_1 s(n-1)+\ldots+b_{2P}s(n-2P) = 0$ where the $2P$ degree polynomial $B(z) = z^{2P} + b_1 z^{2P-1} + \ldots + b_{2P}$ has all of its roots on the unit circle come in pairs of complex conjugate values. Hence the coefficients of $B(z)$ are real.

An important example is that of a *periodic* signal with period T, sum of sines the frequencies of which are multiples of the fundamental frequency $1/T$. In this case, the roots of $B(z)$ are regularly distributed on the unit circle.

Notice that in any case, the recursive equation associated with the signal depends on the sequence of frequencies of this signal (by way of the coefficients of $B(z)$) but is independent of the sequence of complex amplitudes. Remember that if the roots are strictly inside the unit circle, then the signal is evanescent with a time constant that increases as the root with the highest modulus gets closer to 1. We are now going to see that the recursive equation can be modified by filtering.

Filtering a periodic signal with a linear filter

We know that filtering a sine of frequency f_0 leads to a sine with the same frequency f_0, the amplitude of which is multiplied by $H(f_0)$, where $H(f)$ refers to the filter's complex gain. The same goes for any linear combination of sines. Hence, because a sum of sines obeys the M-th order recursive equation, *the signal obtained after filtering obeys the same order recursive equation.* This result should not be surprising since, as we have just seen, this equation has several solutions that differ only by the amplitudes of each frequential component. And the amplitudes of these components are precisely what is set by the filter's frequency response.

Example 10.3 (Impulse train)

Consider the signal made up of a sequence of periodic impulses with period M and with the same amplitude A:

$$x_0(n) = \begin{cases} A & \text{for } n = 0 \bmod M \\ 0 & \text{otherwise} \end{cases}$$

1. Give the recursive equation verified by $x(n)$.
2. Give the general solution $x(n)$ to the previous recursive equation.
3. $x(n)$ is fed into the input of a filter with the complex gain $H(f)$. Give the expression of the output signal $y(n)$. Is it possible to determine $H(f)$ based on the signal $y(n)$?
4. Show that, when the filter is an all-pole filter with the transfer function $1/A(z)$ and $M \gg 1$, $A(z)$ can be estimated using the least squares method without having to estimate M. You can use the `xtoa` function written previously (see page 330). Write a program that checks the result.

HINT:

1. The signal $x(n)$ verifies the recursive equation $x(n) - x(n-M) = 0$.
2. To find the general solution, we must first solve the characteristic equation $z^M - 1 = 0$, the roots of which are $\zeta_k = \exp(2j\pi k/M)$, $k = 0, \ldots, M-1$. The general solution is then given by:

   $$x(n) = \sum_{k=0}^{M-1} \alpha_k \exp(2j\pi kn/M)$$

   where the α_k are M arbitrary complex constants. One example is the impulse sequence $x_0(n)$ for which $\alpha_k = A/M$. We end up with the same identity as 2.33.
3. Based on the complex exponential filtering property and because of linearity, we have:

   $$y(n) = \sum_{k=1}^{M} \alpha_k H(k/M) \exp(2j\pi kn/M) = \sum_{k=1}^{M} \beta_k \exp(2j\pi kn/M)$$

Hence $y(n)$ also verifies $y(n) - y(n-M) = 0$.

Theoretically, without any further information, we cannot find $H(f)$ for any f based on the signal $y(n)$, because the values of $H(f)$ of the type $H(k/M)$ are found in the expression of $y(n)$. However, if M is large, the function $H(f)$ will be sampled at a high rate, and under certain regularity conditions, we can find a unique reconstruction of $H(f)$. An example of a constraint is for the sequence $h(n)$, the inverse DTFT of $H(f)$, to have a duration shorter than M. Another example is to have an all pole filter with the P-th order transfer function $1/A(z)$. In that case, the polynomial $A(z)$ is completely determined once the P frequency points are known, and then we only need to have $M > P$.

4. Since the filter has the P-th order transfer function $1/A(z)$, we can write:

$$y(n) + a_1 y(n-1) + \cdots + a_P y(n-P) = x(n)$$

where $x(n)$ is the sequence of impulses with the amplitude A and the period M. Based on a sequence of observations $y(1), \ldots, y(N)$, and by stacking the recursive equations, we get the matrix expression:

$$\begin{cases} y(P+1) + a_1 y(P) + \cdots + a_P y(1) = x(P+1) \\ y(P) + a_1 y(P-1) + \cdots + a_P y(0) = x(P) \\ \vdots \\ y(N) + a_1 y(N-1) + \cdots + a_P y(N-P) = x(N) \end{cases}$$

This set of expressions can be written as follows:

$$\begin{bmatrix} \mathbf{y} & \mathbf{Y} \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{a} \end{bmatrix} = \mathbf{y} + \mathbf{Y}\mathbf{a} = A\mathbf{i}$$

and can be rewritten as:

$$\mathbf{y} = -\mathbf{Y}\mathbf{a} + A\mathbf{i} \tag{10.25}$$

where $\mathbf{Y}$ is the Toeplitz matrix the first line of which is $[y(P) \ldots y(1)]$, and the first column of which is $[y(P) \ldots y(N-1)]$, $\mathbf{y} = [y(P+1) \ldots y(N)]^T$, $\mathbf{a} = [a_1 \ldots a_P]^T$, and where $\mathbf{i}$ is a periodic eigenvector comprised of a 1 followed by $(M-1)$ zeros. Let us now assume that M is much greater than 1. The vector $\mathbf{i}$ contains almost nothing but zeros, and $\mathbf{a}$ can be estimated using a "least squares" type approach by minimizing, with respect to $\mathbf{a}$, the norm $J(\mathbf{a}) = \|\mathbf{y} - (-\mathbf{Y}\mathbf{a})\|^2 = (\mathbf{y} + \mathbf{Y}\mathbf{a})^T(\mathbf{y} + \mathbf{Y}\mathbf{a})$. If we set the gradient of $J(\mathbf{a})$ to zero, we have:

$$\frac{\partial J(\mathbf{a})}{\partial \mathbf{a}} = 2\mathbf{Y}^T(\mathbf{y} + \mathbf{Y}\mathbf{a}) = 2(\mathbf{Y}^T\mathbf{y} + \mathbf{Y}^T\mathbf{Y}\mathbf{a}) = 0$$

leading to the estimate:

$$\hat{\mathbf{a}} = -(\mathbf{Y}^T\mathbf{Y})^{-1}\mathbf{Y}^T\mathbf{y} = -(\mathbf{Y}^T\mathbf{Y}/N)^{-1}(\mathbf{Y}^T\mathbf{y}/N) \tag{10.26}$$

The matrix $(\mathbf{Y}^T\mathbf{Y})/N$ and the vector $(\mathbf{Y}^T\mathbf{y}/N)$ can be seen as elements constructed from the covariances. Hence expression 10.26 is similar to the equation $\mathbf{a} = -\mathbf{R}^{-1}\mathbf{r}$ derived from the Yule-Walker equations 8.58, which relate the parameters of an AR process to the covariance coefficients. Thus, we can use the `xtoa` function, the advantage of which is to provide us with a stable filter. Type:

```
%===== FILTRAIN.M
clear all, close all
Fs=8000; F0=120; % F0 frequency in Hz
M=round(Fs/F0);  % Period in sample number
%===== For "ideal" pulse P=1
%===== In all cases, P must be much smaller than M
P=1; pulse=ones(P,1);
% An other pulseshape:[0.2;0.7;1;0.3;0.1];
nbpulses=20; lx=nbpulses*M; xs=zeros(lx,1);
for tt=0:nbpulses-1
    ii=tt*M+1;iifin=ii+P-1;
    xs(ii:iifin)=pulse;
end
K=10; % Try any shift
xs=[zeros(K,1);xs]; lx=length(xs);
xs=xs*sqrt(lx)/sum(xs); Px=xs'*xs/lx;
subplot(411); plot((1:lx)/Fs,xs)
nfft=2^nextpow2(lx); freq=Fs*(0:nfft-1)/nfft;
Xf=abs(fft(xs,nfft))/sqrt(lx);
subplot(412); plot(freq,Xf); set(gca,'xlim',[0 Fs/2]);
%===== Filtering
aa=[1;-1.6;0.9]; Hf=1./abs(fft(aa,nfft));
ys=filter(1,aa,xs); Yf=abs(fft(ys,nfft))/sqrt(lx);
subplot(413); plot(freq,Yf);
hold on; plot(freq, Hf,'r'); hold off;
set(gca,'xlim',[0 Fs/2]);
[aae sse]=xtoa(ys,2); [aa aae],[Px sse]
subplot(414); hatxs=filter(aae,1,ys); plot((1:lx)/Fs,hatxs)
```

The advantage of this method is that it requires neither the estimate of the period M nor the estimates of the phases corresponding to the precise times when $x(n)$ is equal to 1.

■

10.3.2 The Prony method

The least squares method used in paragraph 10.1 uses a model for the signal. We already explained in paragraph 10.1 that this leads us to the difficult task of finding the maximum of a multivariable function. The periodogram is another method that approximates the previous calculation. Its advantage is that it only requires the search for the M maxima of a single-variable function. It is, however, limited when it comes to frequency resolution. We know that the order of magnitude for this limit is $2/N$, where N is the duration of the observation.

We are now going to discuss a method invented in the 18th century by the Baron of Prony, a method that enhances the resolution while still having the advantage of requiring only the search for the maximum of a single-variable function.

Exercise 10.1 (The Prony method)
Consider the noised harmonic signal $x(n) = s(n) + w(n)$ where:

$$s(n) = \sum_{k=1}^{P} a_k \cos(2\pi f_k n + \phi_k)$$

and where $w(n)$ is a white, centered, Gaussian noise with the variance σ^2. $\{a_k\}$ refers to a sequence of P unknown amplitudes, assumed to be all different from one another, and ϕ_k refers to a sequence of P unknown phases belonging to $(0, 2\pi)$.

1. Given equation 10.24, how can the sequence of frequencies $f_1, \ldots, f_P$ be determined based on the sequence of values $b_1, \ldots, b_{2P}$?

2. If we assume that $\varepsilon(n) = w(n) + b_1 w(n-1) + \cdots + b_{2P} w(n-2P)$ and add $s(n) + b_1 s(n-1) + \cdots + b_{2P} s(n-2P) = 0$ to the second member, we get $\varepsilon(n) = x(n) + b_1 x(n-1) + \cdots + b_{2P} x(n-2P)$. We are going to try to determine the sequence $b_1, \ldots, b_{2P}$ that minimizes $\sum_n \varepsilon^2(n)$.

   Show that this is equivalent to minimizing, with respect to $\mathbf{b} = [1\ b_1\ \ldots\ b_{2P}]^T$, an expression of the type:

   $$\mathbf{b}^T(\mathbf{D}^T\mathbf{D})\mathbf{b}$$

   where $\mathbf{D}$ is a matrix obtained from the observations $x(0), \ldots, x(N-1)$.

   Using the Lagrange multiplier method (see page 382 and the Capon method), find the expression of $\mathbf{b}$ and therefore the frequencies f_k.

3. Write a program that implements the Prony method. Study this method's performances by choosing for example the test signal $\mathbf{x}$ generated by the program:

```
%===== SIGNALTEST.M
N=25; Am=[2 1.5 1]; F=[0.2 0.225 0.3];
s=Am*cos(2*pi*F'*(0:N-1));
SNR=40; sigma2= (s*s'/N)/(10 ^(SNR/10));
x=s+sqrt(sigma2)*randn(1,N);
```

COMMENTS:

1. The recursive equation $s(n) + b_1 s(n-1) + \cdots + b_P s(n-P) = 0$, given in property 10.1, should be assimilated to equation 8.53 (Chapter 8) which defines an AR-P random process with its second member equal to zero and with its poles on the unit circle. AR identification is known for usually behaving very badly with noised observations. This is also the case of the Prony method, which provides good results only when the noise is low.

2. The Prony method can also be applied to signals that are sums of damped sines of the type:

$$\begin{aligned} s(n) &= \sum_{k=1}^{P} A_k \rho_k^n \cos(2\pi f_k n + \phi_k) \\ &= \sum_{k=1}^{P} \rho_k^n \frac{A_k}{2} \left(e^{j\phi_k} \exp(2j\pi f_k n) + e^{-j\phi_k} \exp(-2j\pi f_k n) \right) \end{aligned}$$

   It can easily be proven that $s(n)$ verifies the recursive equation $s(n) + b_1 s(n-1) + \cdots + b_{2P} s(n-2P) = 0$, where the polynomial $B(z) = z^{2P} + b_1 z^{2P-1} + \cdots + b_{2P}$ has $2P$ complex conjugate roots given by $z_k = \rho_k \exp(\pm 2j\pi f_k)$. The previous algorithm makes it possible to calculate the b_k, then the roots z_k and therefore the frequencies f_k and the damping coefficients ρ_k.

Exercise 10.2 (The Pisarenko method)

Consider the real, discrete-time observation $x(n) = s(n;\theta) + b(n)$, sum of a centered, WSS, random sequence $s(n;\theta)$, that represents the signal with the vector parameter θ we wish to estimate, and of a noise $b(n)$ assumed to be WSS, centered, white, with the variance σ^2, and uncorrelated with $s(n;\theta)$.

Let $R_{ss}(k) = \mathbb{E}\{s(n+k;\theta)s(n;\theta)\}$ be the autocovariance function of $s(n;\theta)$, and let $\mathbf{R}_s$, $\mathbf{R}_b$ and $\mathbf{R}_x$ be the $(M \times M)$ covariance matrices of the random vectors obtained by stacking M consecutive times of $s(n;\theta)$, $b(n)$ and $x(n)$ respectively. Hence, because $s(n;\theta)$ is centered, $\mathbf{R}_s$ can be written:

$$\mathbf{R}_s = \begin{bmatrix} R_{ss}(0) & \cdots & R_{ss}(M-1) \\ \vdots & \ddots & \vdots \\ R_{ss}(-M+1) & \cdots & R_{ss}(0) \end{bmatrix}$$

Because the sequence $R_{ss}(k)$ is even, the matrix $\mathbf{R}_s$ is symmetrical. As for the noise $b(n)$ assumed to be white, we have $\mathbf{R}_b = \sigma^2\mathbf{I}$ where $\mathbf{I}$ refers to the size M identity matrix.

1. Show that $\mathbf{R}_x = \mathbf{R}_s + \sigma^2\mathbf{I}$.

2. The rank of the matrix $\mathbf{R}_s$ is assumed to be $R = (M - G)$ and let $\{\lambda_1, \ldots, \lambda_{M-G}\}$ be the non-zero (hence strictly positive) eigenvalues of $\mathbf{R}_s$. Use this to show that the eigenvalues μ_n of $\mathbf{R}_x$ can be arranged in the following order:

$$\mu_1 \geq \mu_2 \geq \cdots \geq \mu_{M-G} \geq \mu_{M-G+1} = \mu_{M-G+2} = \cdots = \mu_M = \sigma^2$$

3. Let us assume the random signal is of the type:

$$s(n; A, f_0) = A\cos(2\pi f_0 n + \Phi)$$

   where A is an unknown positive amplitude, f_0 is an unknown frequency belonging to $(0, 1/2)$ and Φ is a random variable with a probability distribution uniformly distributed on $(-\pi, +\pi)$ and assumed to be independent of $b(n)$.

   Determine $\mathbb{E}\{s(n)\}$ and $\mathbb{E}\{s(n+k)s(n)\}$. By using the fact that any sine is the solution to a recursive equation with its second member equal to zero, show that the (3×3) matrix $\mathbf{R}_s$, usually has a rank equal to 2. This means that there is a non-zero vector of the type $\mathbf{a} = [1 \quad -2z_0 \quad 0]^T$ such that $\mathbf{R}_s\mathbf{a} = 0$.

4. Generalize the previous result and infer a method for estimating f_0 based on $\mathbf{R}_x$.

5. Evaluate this method's performance using the test signal $\mathbf{x}$ generated by the program `signaltest.m`, and compare the results to those obtained with the Prony method.

```
%===== SIGNALTEST.M
N=25; Am=[2 1.5 1]; F=[0.2 0.225 0.3];
s=Am*cos(2*pi*F'*(0:N-1));
SNR=40; sigma2= (s*s'/N)/(10 ^(SNR/10));
x=s+sqrt(sigma2)*randn(1,N);
```

The Pisarenko method can be generalized and leads to the MUSIC algorithm (for MUltiple SIgnal Classification) presented in paragraph 10.3.3.

10.3.3 The MUSIC algorithm

In this paragraph, we are going to present a method that belongs to the category of what are called subspace methods, and that will provide us with a way of estimating the frequencies of a harmonic mixture by finding the P minima of a single-variable function. The algorithm, the acronym of which is *MUSIC*, for *MUltiple SIgnal Characterization*, works better than the DTFT when the frequency differences are much smaller than the inverse of the number of observed points (resolution limit of Fourier). Furthermore, it can be applied to a broader observation model than that of sines corrupted by noise.

Based on the example of a sum of P real sines corrupted by white noise, we are going to end up with equation 10.29 responsible, because of how it is written, for the notable properties of the covariance matrix.

Sum of P real sines corrupted by white noise

Consider the real observation $x(n)$, $n \in \{0, \dots, N-1\}$, of the type:

$$x(n) = \sum_{k=1}^{P} a_k \cos(2\pi f_k n) + \sum_{k=1}^{P} b_k \sin(2\pi f_k n) + b(n) \tag{10.27}$$

1. $\{f_k\}$ is a sequence of P frequencies, all of them different from one another, belonging to the interval $(0, 1/2)$;
2. $\{a_k\}$ and $\{b_k\}$ are two sequences of P real values;
3. $b(n)$ is a centered, white noise, with the unknown variance σ^2.

Once the sequence f_k has been estimated, the sequences a_k and b_k can be estimated using the least squares method. Refer to expression 10.8 on page 345.

Let us now develop $s(n+\ell)$. We successively get:

$$\begin{aligned} s(n+\ell) &= \sum_{k=1}^{P} a_k \cos(2\pi n f_k)\cos(2\pi\ell f_k) - \sum_{k=1}^{P} a_k \sin(2\pi n f_k)\sin(2\pi\ell f_k) \\ &\quad + \sum_{k=1}^{P} b_k \cos(2\pi n f_k)\sin(2\pi\ell f_k) + \sum_{k=1}^{P} b_k \sin(2\pi n f_k)\cos(2\pi\ell f_k) \end{aligned}$$

Let $\boldsymbol{\theta} = [\theta_1 \quad \dots \quad \theta_P]^T$ where $\theta_k = 2\pi f_k$ and let:

$$\mathbf{u}_\ell(\boldsymbol{\theta}) = \begin{bmatrix} \cos(2\pi\ell f_1) \\ \dots \\ \cos(2\pi\ell f_P) \\ \sin(2\pi\ell f_1) \\ \dots \\ \sin(2\pi\ell f_P) \end{bmatrix} \quad \text{and } \mathbf{s}(n) = \begin{bmatrix} a_1 \cos(2\pi n f_1) + b_1 \sin(2\pi n f_1) \\ \dots \\ a_P \cos(2\pi n f_P) + b_P \sin(2\pi n f_P) \\ -a_1 \sin(2\pi n f_1) + b_1 \cos(2\pi n f_1) \\ \dots \\ -a_P \sin(2\pi n f_P) + b_P \cos(2\pi n f_P) \end{bmatrix}$$

With these notations, we have:

$$s(n+\ell) = \mathbf{u}_\ell^T(\boldsymbol{\theta})\mathbf{s}(n) \tag{10.28}$$

If we stack M values $s(n+\ell)$ for $\ell = 0, \ldots, M-1$, we can write:

$$\begin{bmatrix} s(n) \\ \vdots \\ s(n+M-1) \end{bmatrix} = \begin{bmatrix} \mathbf{u}_0^T(\boldsymbol{\theta}) \\ \vdots \\ \mathbf{u}_{M-1}^T(\boldsymbol{\theta}) \end{bmatrix} \mathbf{s}(n) = \mathbf{A}(\boldsymbol{\theta})\mathbf{s}(n)$$

Finally, if we add noise, we end up with the observation model:

$$\mathbf{x}(n) = \mathbf{A}(\boldsymbol{\theta})\mathbf{s}(n) + \mathbf{b}(n) \tag{10.29}$$

where we have assumed:

$$\mathbf{x}(n) = [x(n) \quad x(n+1) \quad \cdots \quad x(n+M-1)]^T$$

$$\mathbf{b}(n) = [b(n) \quad b(n+1) \quad \cdots \quad b(n+M-1)]^T$$

and where $\mathbf{A}(\boldsymbol{\theta})$ is an $(M \times 2P)$ matrix with the expression:

$$\mathbf{A}(\boldsymbol{\theta}) = \begin{bmatrix}\mathbf{C}(\boldsymbol{\theta}) & \mathbf{S}(\boldsymbol{\theta})\end{bmatrix} \tag{10.30}$$

$$\text{with} \quad \mathbf{C}(\boldsymbol{\theta}) = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \\ \cos(\ell\theta_1) & \cdots & \cos(\ell\theta_P) \\ \vdots & & \\ \cos((M-1)\theta_1) & \cdots & \cos((M-1)\theta_P) \end{bmatrix}$$

$$\text{and} \quad \mathbf{S}(\boldsymbol{\theta}) = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & & \\ \sin(\ell\theta_1) & \cdots & \sin(\ell\theta_P) \\ \vdots & & \\ \sin((M-1)\theta_1) & \cdots & \sin((M-1)\theta_P) \end{bmatrix}$$

Notice that, in expression 10.29, the observation is of the type "signal plus noise", and that the signal part, that is $\mathbf{A}(\boldsymbol{\theta})\mathbf{s}(n)$, is the product of two terms, one of them, $\mathbf{A}(\boldsymbol{\theta})$, depending only on the paramater $\boldsymbol{\theta}$ we wish to estimate, and the other, $\mathbf{s}(n)$, depending only on n.

Based on a sequence of N observations $x(0), \ldots, x(N-1)$, consider the $(M \times M)$ matrix defined by:

$$\widehat{\mathbf{R}}_N = \frac{1}{N-M+1} \sum_{n=0}^{N-M} \mathbf{x}(n)\mathbf{x}^H(n) \tag{10.31}$$

If we change over to the mathematical expectation on the two sides of 10.31, then use 10.29 and the white noise hypothesis, we get:

$$\mathbb{E}\left\{\widehat{\mathbf{R}}_N\right\} = \frac{1}{N-M+1}\sum_{n=0}^{N-M}\mathbb{E}\left\{\mathbf{x}(n)\mathbf{x}^H(n)\right\} = \mathbf{A}(\boldsymbol{\theta})\mathbf{R}_s\mathbf{A}^H(\boldsymbol{\theta}) + \sigma^2\mathbf{I}_M$$

where $\mathbf{I}_M$ is the $M \times M$ identity matrix and where:

$$\mathbf{R}_s = \frac{1}{N-M+1}\sum_{n=0}^{N-M}\mathbf{s}(n)\mathbf{s}^H(n)$$

is a $(2P \times 2P)$ matrix, hence the mean of $\widehat{\mathbf{R}}_N$ is equal to $\mathbf{A}(\boldsymbol{\theta})\mathbf{R}_s\mathbf{A}^H(\boldsymbol{\theta})+\sigma^2\mathbf{I}_M$. We will assume rather than prove that if N is much greater than M, $\widehat{\mathbf{R}}_N$ is a good estimate of the $M \times M$ matrix defined by:

$$\mathbf{R} = \mathbf{A}(\boldsymbol{\theta})\mathbf{R}_s\mathbf{A}^H(\boldsymbol{\theta}) + \sigma^2\mathbf{I}_M \tag{10.32}$$

The MUSIC algorithm uses the fact that the eigendecomposition of the matrix $\mathbf{R}_0 = \mathbf{A}(\boldsymbol{\theta})\mathbf{R}_s\mathbf{A}^H(\boldsymbol{\theta})$ can be obtained directly from that of $\mathbf{R}$, and hence from that of its estimate $\widehat{\mathbf{R}}_N$. Before we present the MUSIC algorithm, we are going to determine an important property inferred from the general form of the observation model provided by expression 10.29.

General form of the observation model

Consider the size M complex observation model:

$$\mathbf{x}(n) = \underbrace{\mathbf{A}(\boldsymbol{\theta})\mathbf{s}(n)}_{\text{signal}} + \mathbf{b}(n) \tag{10.33}$$

where the $M \times P$ matrix $\mathbf{A}(\boldsymbol{\theta})$ with $M > P$ is of the type:

$$\mathbf{A}(\boldsymbol{\theta}) = [\mathbf{a}(\theta_1)\cdots\mathbf{a}(\theta_P)] \tag{10.34}$$

where θ_k belongs to a scalar domain Θ. Notice that the same function $\mathbf{a}(\theta)$ is used to define the P columns of the matrix $\mathbf{A}(\boldsymbol{\theta})$. This model includes of course the case of a sum of P real sines. This is done simply by decomposing each sine with the frequency f_k as the sum of two complex exponentials with the frequencies $\pm f_k$. In that case, the matrix $\mathbf{A}(\boldsymbol{\theta})$ is given by expression 10.30. We will see another application for this observation model in paragraph 10.3.4.

We now come back to the general model 10.33. The P column vectors of the matrix $\mathbf{A}(\boldsymbol{\theta})$ generate in $\mathbb{C}^M$ a P' dimension subspace, with $P' \leq P$, called the *signal subspace*. Its complementary, dimension $(M - P')$ subspace is called the *noise subspace*.

Let $\mathbf{s}(n)$ be a centered, length P, complex vector, and let $\mathbf{R}_s = \mathbb{E}\{\mathbf{s}(n)\mathbf{s}^H(n)\}$ be its covariance matrix. Let $\mathbf{b}(n)$ be a centered, length M, complex noise such that:

$$\mathbb{E}\{\mathbf{b}(n)\mathbf{b}^H(n)\} = \sigma^2\mathbf{I}_M$$

$\mathbf{s}(n)$ and $\mathbf{b}(n)$ are assumed to be uncorrelated. This means that $\mathbf{x}(n)$ is centered and that its covariance matrix has the expression:

$$\mathbf{R} = \mathbb{E}\{\mathbf{x}(n)\mathbf{x}^H(n)\} = \mathbf{A}(\boldsymbol{\theta})\mathbf{R}_s\mathbf{A}^H(\boldsymbol{\theta}) + \sigma^2\mathbf{I}_M = \mathbf{R}_0 + \sigma^2\mathbf{I}_M \qquad (10.35)$$

where we have assumed $\mathbf{R}_0 = \mathbf{A}(\boldsymbol{\theta})\mathbf{R}_s\mathbf{A}^H(\boldsymbol{\theta})$. The form of expression 10.35 leads to the following properties:

$\mathbf{R}_0$ is a positive matrix with a rank $\leq P$

This is because $\mathbf{R}_s$ is a positive matrix, the rank of which is less than or equal to P. Therefore $\mathbf{A}(\boldsymbol{\theta})\mathbf{R}_s\mathbf{A}(\boldsymbol{\theta})^H$ is itself positive with a rank smaller than or equal to P, since it is generated by the P column vectors of $\mathbf{A}$.

If the ranks of $\mathbf{A}(\boldsymbol{\theta})$ and $\mathbf{R}_s$ are equal to P, in other words if $\mathbf{A}(\boldsymbol{\theta})$ and $\mathbf{R}_s$ are full rank matrices, then $\mathbf{R}_0$ is a full rank matrix.

$\mathbf{R}$ and $\mathbf{R}_0$ have the same eigenvectors

This is because $\mathbf{R}_0$ has $P' \leq P$ strictly positive eigenvalues and $(M - P')$ null eigenvalues. This result is a direct consequence of the previous result. The set of these eigenvectors is an orthonormal basis of $\mathbb{C}^M$.

Let $\mathbf{v}_1, \ldots, \mathbf{v}_{P'}$ be the eigenvectors associated with the strictly positive eigenvalues of $\mathbf{R}_0$. We have $\mathbf{R}_0\mathbf{v}_i = \lambda_i\mathbf{v}_i$. If we multiply equation 10.35 on the right by $\mathbf{v}_i$, we get $\mathbf{R}\mathbf{v}_i = (\sigma^2+\lambda_i)\mathbf{v}_i$. Therefore, $\mu_i = \sigma^2+\lambda_i > \sigma^2$ is an eigenvalue of $\mathbf{R}$ associated with the eigenvector $\mathbf{v}_i$.

Let $\mathbf{g}_1, \ldots, \mathbf{g}_K$, with $K = M - P'$, be the eigenvectors associated with the null eigenvalues of $\mathbf{R}_0$. We have $\mathbf{R}_0\mathbf{g}_i = 0$. If we multiply equation 10.35 on the right by $\mathbf{g}_i$, we get $\mathbf{R}\mathbf{g}_i = \sigma^2\mathbf{g}_i$. Therefore σ^2 is an eigenvalue of $\mathbf{R}$ with multiplicity K, associated with the eigenvectors $\mathbf{g}_1, \ldots, \mathbf{g}_K$.

Remember that $\mathbf{v}_j^H\mathbf{g}_k = 0$ for any pair (j, k), which is a consequence of the orthogonality of the eigendecomposition of a positive matrix.

COMMENT: we saw on page 352 that the periodogram of a sum of sines and of a noise tends to σ^2 for the frequency values that are different from the frequencies of the sine components. This result is similar to the previous one stating that the eigenvalues of the noise subspace are all equal to σ^2. The Fourier transform performs some kind of orthogonal decomposition that approximately separates the space in a signal subspace and a noise subspace. In the noise subspace, each component then has the same power σ^2.

As a conclusion, we have the following theorem.

Theorem 10.1 *Let* $\mathbf{x}(n)$ *be the size* M *complex observation model:*

$$\mathbf{x}(n) = \mathbf{A}(\boldsymbol{\theta})\mathbf{s}(n) + \mathbf{b}(n) \tag{10.36}$$

where the $(M \times P)$ *matrix* $\mathbf{A}(\boldsymbol{\theta})$ *with* $M > P$ *is of the type:*

$$\mathbf{A}(\boldsymbol{\theta}) = [\mathbf{a}(\theta_1) \cdots \mathbf{a}(\theta_P)]$$

$\mathbf{s}(n)$ *is a centered process with the covariance matrix* $\mathbf{R}_s = \mathbb{E}\left\{\mathbf{s}(n)\mathbf{s}^H(n)\right\}$, $\mathbf{b}(n)$ *is a centered white noise with the covariance matrix* $\mathbb{E}\left\{\mathbf{b}(n)\mathbf{b}^H(n)\right\} = \sigma^2\mathbf{I}_M$. $\mathbf{s}(n)$ *and* $\mathbf{b}(n)$ *are assumed to be uncorrelated. If* $\mathbf{R}_0 = \mathbf{A}(\boldsymbol{\theta})\mathbf{R}_s\mathbf{A}^H(\boldsymbol{\theta})$ *then* $\mathbf{x}(n)$ *is centered and its covariance matrix is such that:*

$$\mathbf{R} = \mathbf{R}_0 + \sigma^2\mathbf{I} = \mathbf{V}\Lambda\mathbf{V}^H + \sigma^2\mathbf{G}\mathbf{G}^H \tag{10.37}$$

where the matrix $\mathbf{V}$ *is comprised of* $P' \leq P$ *orthonormal eigenvectors of* $\mathbf{R}_0$ *associated with strictly positive eigenvalues, where* $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_{P'})$ *is the diagonal matrix with these eigenvalues on its diagonal, and where the matrix* $\mathbf{G}$ *is comprised of the* $M - P'$ *unit eigenvectors of* $\mathbf{R}_0$ *associated with null eigenvalues. We have* $\mathbf{V}^H\mathbf{G} = 0$*. Notice that* $\mathbf{G}^H\mathbf{G} = \mathbf{I}$ *and therefore that* $\mathbf{G}\mathbf{G}^H$ *is the orthogonal projector onto the noise subspace.*

Theorem 10.1 implies the following.

Property 10.2 *The subspace generated by the columns of* $\mathbf{V}$ *coincides with the subspace generated by the columns of* $\mathbf{A}(\boldsymbol{\theta})\mathbf{R}_s$*, and both are contained in the space generated by the columns of* $\mathbf{A}(\boldsymbol{\theta})$ *This means that if* P' *refers to the rank of* $\mathbf{R}_s$ *where* $P' \leq P$*, then there is a full rank* $(P \times P')$ *matrix* $\mathbf{T}$ *such that* $\mathbf{V} = \mathbf{A}(\boldsymbol{\theta})\mathbf{T}$.

Estimation based on the noise subspace

Consider the general complex situation presented in theorem 10.1. Starting off with the orthogonality property of $\mathbf{A}(\boldsymbol{\theta})$ with the noise subspace, we are going to construct an estimation of θ_1, ..., θ_P based on the P minima of a *single*-variable function. In the case where $\mathbf{a}(\theta)$ is of the complex exponential type, this function is in the form of a trigonometric polynomial.

If we multiply equation 10.37 on the right by $\mathbf{G}$, we get $\mathbf{R}_0\mathbf{G} = 0$ and therefore:

$$\mathbf{A}(\boldsymbol{\theta})\mathbf{R}_s\mathbf{A}^H(\boldsymbol{\theta})\mathbf{G} = 0 \tag{10.38}$$

The MUSIC estimator searches for the value of $\boldsymbol{\theta}$ such that $\mathbf{A}^H(\boldsymbol{\theta})\mathbf{G} = 0$, which means that 10.38 is true. The converse is true if the $(M \times P)$ matrix $\mathbf{A}(\boldsymbol{\theta})\mathbf{R}_s$ is a full rank matrix.

Remember that for any matrix $\mathbf{M}$, we have the equivalence:

$$\mathbf{M} = 0 \Longleftrightarrow \mathrm{Tr}(\mathbf{M}\mathbf{M}^H) = 0 \tag{10.39}$$

We simply have to notice that $\mathrm{Tr}(\mathbf{M}\mathbf{M}^H) = \sum_i \sum_k |m_{ik}|^2$, where m_{ik} refers to the generic element of $\mathbf{M}$.

Using 10.39 then leads us to the following expression for the MUSIC estimator of the $\boldsymbol{\theta}$ parameter.

Property 10.3 (MUSIC estimator) *The MUSIC estimator of the $\boldsymbol{\theta}$ parameter associated with the model 10.33 is:*

$$\hat{\boldsymbol{\theta}}_{\mathrm{MUSIC}} = \arg \min_{\boldsymbol{\theta} \in \Theta^P} \mathrm{Tr}\left(\mathbf{A}(\boldsymbol{\theta})\mathbf{A}^H(\boldsymbol{\theta})\mathbf{G}\mathbf{G}^H\right) \tag{10.40}$$

where $\mathbf{G}\mathbf{G}^H$ is the orthogonal projector onto the noise subspace obtained from the decomposition of the covariance matrix estimate.

If we replace 10.34 in expression 10.40, we get:

$$\mathrm{Tr}\left(\mathbf{A}(\boldsymbol{\theta})\mathbf{A}^H(\boldsymbol{\theta})\mathbf{G}\mathbf{G}^H\right) = \sum_{k=1}^{P} \mathrm{Tr}(\mathbf{a}(\theta_k)\mathbf{a}^H(\theta_k)\mathbf{G}\mathbf{G}^H) = \sum_{k=1}^{P} \mathbf{a}^H(\theta_k)\mathbf{G}\mathbf{G}^H\mathbf{a}(\theta_k)$$

Because $\mathbf{G}\mathbf{G}^H$ is a positive matrix, the minimization is equivalent to finding the P arguments of the P minima that are closest to 0 of the *single*-variable function:

$$J(\theta) = \mathbf{a}(\theta)^H\mathbf{G}\mathbf{G}^H\mathbf{a}(\theta) \tag{10.41}$$

Numerical computation of the P minima

The simplest and broadest method for finding the minima of $J(\theta)$ consists of calculating $J(\theta)$ on a set of values θ sampled on a fine grid (see also exercise 10.3). The minima are identified by typing:

```
idxMin=find(diff(sign(diff(J)))==2);
```

This method will be used in the FFT-MUSIC algorithm.

We are now going to see methods for which $\mathbf{a}(\theta)$ is comprised of complex exponentials, which means we can use the FFT.

Case where the components are complex exponentials

Consider the particular case where the function can be written:

$$\mathbf{a}(\theta) = \begin{bmatrix} 1 & e^{j\theta} & \cdots & e^{j(M-1)\theta} \end{bmatrix}^T$$

This includes the case defined by expression 10.30. All we have to do is group together the two columns $\mathbf{c}(\theta_k)$ and $\mathbf{s}(\theta_k)$ obtained from the sequences $\cos(\ell\theta_k)$ and $\sin(\ell\theta_k)$, and then set $\mathbf{a}(\theta_k) = \mathbf{c}(\theta_k) + j\mathbf{s}(\theta_k)$.

According to 10.41, an estimation of $\theta_1, \ldots, \theta_P$ is obtained by determining the P minima of the single-variable function:

$$Q(e^{j\theta}) = \mathbf{a}^H(\theta)\mathbf{G}\mathbf{G}^H\mathbf{a}(\theta)$$

Notice that $Q(e^{j\theta})$ can also be interpreted as the value, calculated on the unit circle, of the polynomial:

$$Q(z) = \mathbf{a}_z^H(1/z^*)\mathbf{G}\mathbf{G}^H\mathbf{a}_z(z) \tag{10.42}$$

where $\mathbf{a}_z(z) = \begin{bmatrix}1 & z & \ldots & z^{M-1}\end{bmatrix}^T$ and where $z = e^{j\theta}$. With this notation, we then have $\mathbf{a}_z^H(1/z^*) = \begin{bmatrix}1 & z^{-1} & \ldots & z^{-(M-1)}\end{bmatrix}$ and:

$$Q(z) = z^{-(M-1)}\begin{bmatrix}z^{M-1} & \cdots & z & 1\end{bmatrix}\mathbf{G}\mathbf{G}^H\begin{bmatrix}1\\ z\\ \vdots\\ z^{M-1}\end{bmatrix} = z^{-(M-1)}\tilde{Q}(z) \tag{10.43}$$

$\tilde{Q}(z)$ is a $2(M-1)$ degree polynomial in z the roots of which come in pairs, since, by construction, if z_0 is a root, then $1/z_0^*$ is a root.

Calculation of the coefficients of the polynomial $\tilde{Q}(z)$

Let $\mathbf{P} = \mathbf{G}\mathbf{G}^H$ be the matrix with its generating element given by:

$$p_{ij} = \sum_{k=1}^{K} g_{ik}g_{jk}^* \text{ for } i,j = 1,\ldots,M$$

where g_{ik} is the i-th component of the k-th column vector of $\mathbf{G} = [\mathbf{g}_1 \ldots \mathbf{g}_k \ldots \mathbf{g}_K]$. The coefficients q_d of the polynomial $\tilde{Q}(z) = \sum_{d=0}^{2M-2} q_d z^d$ are obtained from the relation 10.43, which is written:

$$\begin{cases} q_d = \sum_{k=0}^{d} p_{M-d+k,k+1} & \text{for} \quad d = 0,\ldots,(M-1) \\ q_d = q_{2M-d-2}^* & \text{with} \quad d = M,\ldots,(2M-2) \end{cases}$$

As you can see, the coefficients q_d are calculated as the sum of the diagonal terms of the matrix $\mathbf{G}\mathbf{G}^H$ according to the diagram in Figure 10.7.

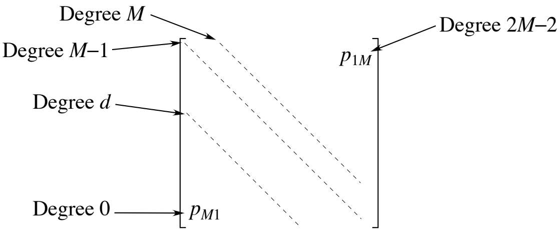


Figure 10.7 – *Calculation of the coefficients of* $\tilde{Q}(z)$ *based on* $\mathbf{P} = \mathbf{G}\mathbf{G}^H$

Implementation of the MUSIC algorithm

The following sums up the MUSIC algorithm in the case of an observation that is the sum of P complex exponential components. In the particular case where the signal $x(n)$ is the sum of P real sines, all we have to do is apply the algorithm by considering $x(n)$ as a linear combination of complex exponentials.

1. *Choose* $M > P$.
2. *Calculate:*

$$\hat{R}_N = \frac{1}{N-M+1} \sum_{n=0}^{N-M} \mathbf{x}(n)\mathbf{x}^H(n)$$

$$\textit{with } \mathbf{x}(n) = [x(n) \quad \cdots \quad x(n+M-1)]^H$$

3. *Calculate the eigendecomposition of* $\hat{R}_N$. *Use it to find the* $(M \times (M-P))$ *matrix* $\mathbf{G}$, *constructed from the* $(M-P)$ *eigenvectors associated with the* $(M-P)$ *smallest eigenvalues. Calculate the* $(M \times M)$ *matrix* $\mathbf{G}\mathbf{G}^H$.
4. *Use the previous result to find the coefficients of the polynomial (equation 10.43):*

$$\tilde{Q}(z) = \begin{bmatrix} z^{M-1} & \cdots & z & 1 \end{bmatrix} \mathbf{G}\mathbf{G}^H \begin{bmatrix} 1 & z & \cdots & z^{M-1} \end{bmatrix}^T$$

Once the polynomial $\tilde{Q}(z)$ is obtained, the estimation of the P values $f_k = \theta_k/2\pi$ can then be achieved, among other possibilities:

1. by calculating the $2(M-1)$ roots of $\tilde{Q}(z)$, then keeping the P stable roots that are closest to the unit circle. This is called the *root-music* method;

2. or by finding the P minima of $\tilde{Q}(e^{j\theta})$. This is achieved simply by calculating, with the help of the **fft** function, $\tilde{Q}(e^{j\theta})$ for $\theta = 2\pi k/L$, where $k \in \{0, \dots, L-1\}$. This is called the *fft-music* method.

The two methods lead basically to the same values if the roots of the polynomial $\tilde{Q}(z)$ are very close to the unit circle. Remember that they would be on the unit circle if the signal were a perfect mixture of exponentials without noise. The FFT method then has a small advantage, because root finding algorithms often require more computation time. However, in the presence of significant noise, it would seem the root finding method is better suited.

Performances depend on the choice of M. What should be remembered is that M has to tend to infinity when N tends to infinity, but not as fast as N, which is the case for example for $M = N^{\gamma}$, with $\gamma < 1$, such as $\gamma = 4/5$, or $\gamma = 2/3$.

ROOT-MUSIC The `music(xm,p)` function given below implements the MUSIC algorithm for the estimation of the frequencies of a sum of P complex exponentials by using a root finding method for the polynomial $\tilde{Q}(z)$. It first calculates $\hat{R}_n$ based on the expression 10.31 where we chose $M = N^{4/5}$ (try also for example $M = N^{2/3}$). Then it calculates the vectors of the noise subspace with the use of the MATLAB® function `svd`[1] and uses the result to find the coefficients of the polynomial $\tilde{Q}(z)$ (given by 10.43). Finally, the function returns the roots of $\tilde{Q}(z)$ using the MATLAB® function `roots`. Based on the values of these roots, it can then estimate the frequencies $f_1, \dots, f_P$ using the `angle` function.

```
function [z]=music(xm,p)
%%================================================%
%% MUSIC: Estimation of the roots of a polynomial %
%% for p complex exponentials in a white noise    %
%% SYNOPSIS: [z]=MUSIC(xm,p)                      %
%%     xm = Complex observations                  %
%%     p  = Number of complex exponentials        %
%%     z  = Estimation of the p complex roots     %
%%          corresponding to the p frequencies    %
%%================================================%
xm=xm(:); xm=xm-mean(xm); N=length(xm);
%===== M must be > p
M=fix(N^(4/5));
R2x=zeros(M,M);
```

[1] You can also use the MATLAB® function `eig`, since the matrix we are dealing with is square, positive, and therefore its singular value decomposition (`svd` function) and its eigendecomposition (`eigen values`) coincide. However, the `eig` function does not sort the eigenvalues, and must be followed by the MATLAB® function `sort`.

```
for ii=1:N-M+1
    idb=ii; ifn=idb+M-1;
    C2x=xm(idb:ifn)*xm(idb:ifn)';
    R2x=R2x+C2x;
end
R2x=R2x/(N-M+1); [u1 d1 v1]=svd(R2x);
GG=v1(:,p+1:M); PP=GG*GG'; QQ=zeros(2*M-1,1);
%===== Constructing Q(z)
for d=0:M-1
    for k=0:d
        QQ(d+1) = QQ(d+1) + PP(M-d+k,k+1);
    end
end
QQ(M+1:2*M-1)=conj(QQ(M-1:-1:1));
zz=roots(QQ(2*M-1:-1:1));
%===== The roots with moduli > 1
%      and with non-zero imaginary parts
v=find(abs(zz)<=1 & imag(zz)~=0);
z=sort(zz(v)); mr=length(z); z=z(mr-p+1:mr);
return
```

This function can be used for signals containing P real sines, simply by considering $2P$ complex exponentials. The function returns a sequence of complex conjugate values (see example 10.4).

FFT-MUSIC The musicFFT(xm,p) function implements the MUSIC algorithm for the estimation of the frequencies of a sum of P complex exponentials by searching for the minima of the polynomial on the unit circle. It first calculates $\hat{R}_n$ based on the expression (10.31) where we chose $M = N^{4/5}$. Then it uses the result to find the coefficients of the polynomial $\tilde{Q}(z)$ (given by 10.43). Finally, it returns the P values of z on the unit circle that minimize $\tilde{Q}(z)$. This is achieved by the FFT computation of Lfft values of the polynomial $\tilde{Q}(z)$. Then the P minima are found first by determining the sign of the derivative between two consecutive values with the use of the command dQQfft=filter([-1 1],1,QQfft) (you can also use the diff function which returns one less value). When the sign of the derivative goes from -1 to $+1$, it means we just went by a minimum. To run this test, the filter function is used once more by executing dsdQQfft=filter([1 -1],1,sdQQfft) and comparing the result with the value $(+1) - (-1) = 2$.

The choice of Lfft modifies the accuracy of the calculation of the obtained minima. We chose Lfft=16*1024 in the function function musicFFT.m. Restricting the search to the minima by calculating more values of $\tilde{Q}(z)$ could save you some time, but only in the neighborhoods of the obtained values.

```
function [zout,QQfft]=musicFFT(xm,p)
```

```
%%===================================================%
%% MUSIC: Estimation of the frequencies in a mixture %
%% of p complex exponentials in a white noise        %
%% SYNOPSIS: [zout,QQfft]=MUSICFFT(xm,p)             %
%%     xm    = Complex observation sequences         %
%%     p     = Number of complex exponentials        %
%%     zout  = Estimation of the p complex roots     %
%%             on the unit circle                    %
%%===================================================%
xm=xm(:); xm=xm-mean(xm); N=length(xm);
%===== M must be > p
M=fix(N^(4/5)); R2x=zeros(M,M);
for ii=1:N-M+1
    idb=ii; ifn=idb+M-1;
    C2x=xm(idb:ifn)*xm(idb:ifn)'; R2x=R2x+C2x;
end
R2x=R2x/(N-M+1); [u1 d1 v1]=svd(R2x);
GG=v1(:,p+1:M); PP=GG*GG'; QQ=zeros(2*M-1,1);
%===== Constructing Q(z)
for d=0:M-1
    for k=0:d
        QQ(d+1) = QQ(d+1) + PP(M-d+k,k+1);
    end
end
QQ(M+1:2*M-1)=conj(QQ(M-1:-1:1)); QQ=QQ(2*M-1:-1:1);
Lfft=16*1024; QQfft=abs(fft(QQ,Lfft));
%===== Computation of QQfft(n)-QQfft(n-1)
dQQfft=filter([1 -1],1,QQfft);
%===== If QQfft increases sdQQfft=+1
sdQQfft=sign(dQQfft); dsdQQfft=filter([1 -1],1,sdQQfft);
pos=find(dsdQQfft==2);
[QQfftpos indQQ]=sort(QQfft(pos));
pos=pos(indQQ); pos=pos(1:p);
zout=exp(2*j*pi*(pos-1)/Lfft);
% pos=find(diff(sign(diff(QQfft)))==2);
% [QQfftpos indQQ]=sort(QQfft(pos));
% pos=pos(indQQ); pos=pos(1:p);
% zout=exp(2*j*pi*(pos+1)/Lfft);
return
```

Example 10.4 (Implementation and simulations)

1. Write a program that generates a size $N = 80$ samples of the $P = 2$ sines defined by the values:

f_k	0.2	0.21
a_k	1	0.2
b_k	0	0

with a noise added to it such that the SNR is equal to 20 dB.

2. We wish to evaluate the performances depending on the signal-to-noise ratio. This is achieved by setting $N = 80$, $f_1 = 0.2$, $f_2 = 0.21$ and the amplitude ratio to 0.2. The signal-to-noise ratio varies between 10 and 20 dB. The square deviation between the estimator and the real value is a useful performance indicator. In practice, the analytical expression of the result is impossible to obtain, which is why simulations are done by performing a large number of trials. This is called a *Monte-Carlo* simulation. Write a program that conducts such a simulation based on 300 trials.

HINT:

1. Type:

```
%===== APPLMUSIC.M
clear
nfft=1024; freq=(0:nfft-1)/nfft;
SNRdB=20;             % SNR
N=80; tps=(0:N-1);    % N = number of samples
fq=[0.2 0.21]; p=length(fq); alpha=[1 0.2];
%===== Signal
sig=alpha * cos(2*pi*fq'*tps);
vseff=std(sig);
siggmab=sqrt(10^(-SNRdB/10))*vseff/sqrt(2);
b=siggmab*randn(1,N);
x=sig+b;              % Noised signal
%=====
[racm,QQfft]=musicFFT(x,2*p);              % MUSIC
fqm=angle(racm)/(2*pi);
fqaux=sort(fqm);              % Sort the frequencies
fqmo=fqaux(p+1:2*p); regang=2*pi*fqmo*tps;
RR=[cos(regang)' sin(regang)'];
ab=RR \ x';
alphamo=sqrt(ab(1:p) .^2 + ab(p+1:2*p) .^2);
%===== Displaying the DTFT
subplot(211)
plot(freq,20*log10(abs(fft(x,nfft))/N))
set(gca,'xlim',[0 0.5])
zoom xon; grid
LQ=length(QQfft);
subplot(212); plot([0:LQ-1]/LQ,-20*log10(abs(QQfft)));
set(gca,'xlim',[0 1/2]); grid
%===== Displaying the results
disp(sprintf('SNR : \t %5.4g dB',SNRdB));
disp(sprintf('Number of samples : \t %3i',N));
disp(sprintf('Number of sines : \t %3i',p));
disp('True values :')
disp(sprintf('Freq. = %5.4g\t ampl. = %5.4g\t \n', ...
             [fq;alpha]))
disp('Estimated values :')
disp(sprintf('\t freq. = %5.4g\t ampl. = %5.4g\t \n', ...
            [fqmo';alphamo']))
```

As you can see with the program **applmusic.m**, the DTFT is not able to properly estimate the frequencies f_1 and f_2 whereas the MUSIC algorithm does.

2. **simumusic.m** implements a simulation to evaluate the performances for the measurement of f_1 and f_2 for several values of the signal-to-noise ratio.

   The graph on the left shows the mean square deviation of the estimation of f_1, and the one on the right shows the mean square deviation of the estimation of f_2, for several values of the signal-to-noise ratio. Notice that in both cases, the square deviation "decreases" as the signal-to-noise ratio increases. Furthermore, performances are better for f_1 than they are for f_2, because the amplitude of the sine with the frequency f_2 is much smaller than the one associated with f_1.

```
%===== SIMUMUSIC.M
clear;
N=80; tps=(0:N-1);      % N = number of samples
fq=[0.2 0.21]; alpha=[1 0.2]; p=length(fq);
%===== Signal
sig=alpha * cos(2*pi*fq'*tps);
vseff=std(sig)/sqrt(2);
%===== SNR
SNR=(10:2:20); lSNR=length(SNR);
eqmfq=zeros(lSNR,p);
L=30;                   % Number of trials
for ii=1:lSNR
    SNRdB=SNR(ii);
    siggmab=sqrt(10^(-SNRdB/10))*vseff;
    fqmo=zeros(p,L); alphamo=zeros(p,L);
    for jj=1:L
        b=siggmab*randn(1,N);
        %==== Noised signal
        x=sig+b; racm=music(x,2*p);
        fqm=angle(racm)/(2*pi); fqaux=sort(fqm);
        fqmo(:,jj)=fqaux(p+1:2*p);
    end
    dfqmo=fqmo-fq'*ones(1,L);
    eqmfq(ii,:)=std(dfqmo');
end
```

```
%===== Displaying results
subplot(121); plot(SNR,eqmfq(:,1),'o'); grid
hold on; plot(SNR,eqmfq(:,1)); hold off
subplot(122); plot(SNR,eqmfq(:,2),'o'); grid
hold on; plot(SNR,eqmfq(:,2)); hold off
```

You can also conduct the simulation for the function `musicFFT.m`. ■

10.3.4 Introduction to array processing

Figure 10.8 shows a uniform linear array (ULA) comprising $M = 3$ sensors assumed to be identical, isotropic and separated by the distance d. A source located in the direction φ sends a wave with the carrier frequency F_0. This source is assumed to be far enough for the wave's front to be considered parallel lines.

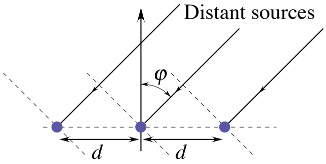


Figure 10.8 – *Uniform linear array with* 3 *sensors and* $P = 1$ *distant source assumed to be far away. The dashed lines represent the equiphase lines*

Under these conditions, if $y_1(t) = x_1(t)e^{2j\pi F_0 t}$ where $x_1(t)$ represents the complex envelop of the signal $y_1(t)$ received by sensor 1, then the signal $y_2(t)$ received by sensor 2, located at a distance d from sensor 1, has the expression $y_2(t) = y_1(t-\tau) = x_1(t-\tau)e^{2j\pi F_0(t-\tau)}$, where the delay $\tau = d\sin\varphi/c$ and where $\varphi \in (-\pi/2, \pi/2)$. c refers to the propagation speed. In radiocommunications, $c = 3 \times 10^8$ m/s.

Narrow band hypothesis

Let us assume that $B_s \ll F_0$, where B_s refers to the frequency band of the signal $x_1(t)$. Then the signal $y_1(t)$ is said to be a narrow band signal around F_0. In that case, if we use $F_0 = c/\lambda_0$ where λ_0 refers to the wavelength and $\tau = d\sin\varphi/c$, then $B_s\tau \ll F_0 d/c = d/\lambda_0$.

When $c = 3 \times 10^8$, the value of $f_0 d/c$ can be close to 1, meaning that the condition $b_s\tau \ll 1$ is met, and hence that $x_1(t)$ varies little during the time τ. We can then infer that $x_1(t-\tau) \approx x_1(t)$. As a conclusion, when the signals are narrow band and when the propagation speed is very high:

$$y_2(t) \approx x_1(t)e^{-2j\pi F_0\tau}e^{2j\pi F_0 t} \quad \text{then} \quad x_2(t) = x_1(t)e^{-2j\pi F_0\tau} \qquad (10.44)$$

where $x_2(t)$ represents the complex envelop of $y_2(t)$.

This result is no longer true for acoustic propagation (SONAR type propagation), for which the value of c is too small, meaning that the value of $F_0 d/c$ is always high. In that case, it is not always possible to have $B_s\tau \ll 1$. From now on, we will assume that the conditions are those of the first case, meaning that the approximation $x_1(t-\tau) \approx x_1(t)$ is valid.

From now on, $\mathbf{x}(n)$, $n \in \mathbb{Z}$, denotes the received signal sampled at the frequency F_s and $f_0 = F_0/F_s$. Expressions 10.44 are true for two sensors, and can easily be extended to a set of M sensors, which leads to:

$$\mathbf{x}(n) = [x_1(n), \ldots, x_M(n)]^T = \mathbf{a}(\varphi) x_1(n)$$

where:

$$\mathbf{a}(\varphi) = \begin{bmatrix} 1 & e^{-j\theta(\varphi)} & \ldots & e^{-j(M-1)\theta(\varphi)} \end{bmatrix}^T \tag{10.45}$$

with:

$$\theta(\varphi) = 2\pi \frac{d}{\lambda_0} \sin(\varphi) \tag{10.46}$$

An antenna is said to be unambiguous if $\varphi = \varphi' \Leftrightarrow \theta = \theta' (\bmod 2\pi)$. It is simple to show that the ULA is unambiguous if:

$$d \leq \frac{\lambda_0}{2}$$

that is to say that the distance between two sensors must be smaller than half the wavelength.

If we now assume that there are P sources, located in P directions $\varphi_1, \ldots, \varphi_P$ of the plane, and that the observation is the sum of these P contributions and of a noise, then we can write:

$$\begin{array}{ccccccc} \mathbf{x}(n) & = & \mathbf{A}(\boldsymbol{\varphi}) & \times & \mathbf{s}(n) & + & \mathbf{b}(n) \\ (M \times 1) & & (M \times P) & & (P \times 1) & & (M \times 1) \end{array} \tag{10.47}$$

where $\mathbf{s}(n) = [x_1^1(n), \ldots, x_1^P(n)]^T$ (envelops on the first captor) and where:

$$\mathbf{A}(\boldsymbol{\varphi}) = \begin{bmatrix} \mathbf{a}(\varphi_1) & \mathbf{a}(\varphi_2) & \ldots & \mathbf{a}(\varphi_P) \end{bmatrix}$$

The vectors $\mathbf{a}(\varphi_k)$ are called the *steering vectors.* $\mathbf{b}(n)$ is an $(M \times 1)$ noise vector. The noise is assumed to be centered and white. We then have $\mathbb{E}\{\mathbf{b}(n)\} = \mathbf{0}$ and:

$$\mathbb{E}\{\mathbf{b}(n)\mathbf{b}^H(k)\} = \delta_{n,k}\sigma^2 \mathbf{I}_M$$

Notice that the signal model provided by equation 10.47 is identical to the one given by equation 10.33 where θ is given by 10.46.

We will not extensively discuss the problem of determining the P arrival directions based on the observation of N shots $\{\mathbf{x}(1), \ldots, \mathbf{x}(N)\}$.

We will merely give a few answers.

Classical beamforming

A narrow band beamformer is basically a delay-and-sum processing to steer a beam in a particular direction: the beamformer combines the signals sensed by the antenna array with complex weights $\mathbf{w}$. From equation 10.47, if:

$$\mathbf{x}(n) = \sum_{k=1}^{P} \mathbf{a}(\varphi_k) s_k + \mathbf{b}(n)$$

denotes the observed signal, a beamformer gives the signal $y(n) = \mathbf{w}^H \mathbf{x}(n)$. For the ULA, the simplest idea is to choose $\mathbf{w} = \mathbf{a}(\varphi_k)$ for the extracting of the source k. Then the output of the so-called classical beamformer is written:

$$\begin{aligned} y(n) &= \mathbf{a}^H(\varphi_k)\mathbf{x}(n) \\ &= \mathbf{a}^H(\varphi_k)\mathbf{a}(\varphi_k)s_k + \textstyle\sum_{p \neq k} \mathbf{a}^H(\varphi_k)\mathbf{a}(\varphi_p)s_p + \mathbf{a}^H(\varphi_k)\mathbf{b}(n) \end{aligned} \quad (10.48)$$

where the first term of 10.48 represents the signal of interest, the second one the interference and the third one the noise. It can be shown that:

$$\mathbf{a}^H(\varphi_k)\mathbf{a}(\varphi_p) = \frac{\sin(M(\theta_k - \theta_p))}{\sin(\theta_k - \theta_p)} e^{j(M-1)(\theta_k - \theta_p)}$$

where $\theta_p = 2d \sin(\varphi_p)/\lambda_0$ (equation 10.46). Then, if

$$\min_{k \neq p} |\theta_k - \theta_p| \gg 1/M \quad (10.49)$$

the term of interference will be close to 0. This condition can also be written:

$$\min_{k \neq p} \frac{Md}{\lambda_0} 2|\sin((\varphi_k - \varphi_p)/2)\cos((\varphi_k + \varphi_p)/2)| \gg 1$$

If we need to distinguish arrival angles separated by e, for small values of e, we can write that $Med/\lambda_0 \gg 1$. For $e = 1/10$ radian and for $d = \lambda_0/2$, we have to set $M \gg 20$.

Classical beamforming approach may also be used to estimate, from N observations, the P angles of arrival φ_p by searching the P maxima w.r.t. φ of the function:

$$S_{\mathrm{B}}(\varphi) = \frac{1}{N} \sum_{n=1}^{N} |\mathbf{a}^H(\varphi)\mathbf{x}(n)|^2 = \mathbf{a}^H(\varphi)\hat{\mathbf{R}}_N \mathbf{a}(\varphi) \quad (10.50)$$

where $\hat{\mathbf{R}}_N = N^{-1} \sum_{n=1}^{N} \mathbf{x}(n)\mathbf{x}^H(n)$.

In the case where condition 10.49 is not well satisfied, the performances of this method become mediocre. Different methods can be used, such as the MUSIC algorithm, seen in paragraph 10.3.3.

The Capon method

In order to enhance the resolution obtained by the classic approach, Capon suggests in [19] searching for the weighting vector $\mathbf{w}$ which, on the one hand, minimizes:

$$\mathbb{E}\{|\mathbf{w}^H\mathbf{x}(n)|^2\} = \mathbf{w}^H\mathbf{R}\mathbf{w}$$

where $\mathbf{R} = \mathbb{E}\{\mathbf{x}(n)\mathbf{x}^H(n)\}$ and, on the other, satisfies the constraint $\mathbf{w}^H\mathbf{a}(\varphi_k) = 1$. The idea is to perform a weighted sum of the observations in order to cancel every component, except for the one coming from the direction φ_k for which the gain must be 1. $\mathbf{w}$ therefore performs a spatial filtering that focalizes the antenna on the source located in the direction φ_k.

The solution is obtained using the Lagrange multiplier method, which consists of solving the following equivalent problem:

$$\begin{cases} \min_{\mathbf{w}} \left[\mathbf{w}^H\mathbf{R}\mathbf{w} - \lambda(\mathbf{w}^H\mathbf{a}(\varphi_k) - 1)\right] \\ \mathbf{w}^H\mathbf{a}(\varphi_k) - 1 = 0 \end{cases} \tag{10.51}$$

By setting to zero the derivative with respect to $\mathbf{w}$ of the first expression, we get $\mathbf{R}\mathbf{w} - \lambda\mathbf{a}(\varphi_k) = 0$, which leads to:

$$\mathbf{w} = \lambda\mathbf{R}^{-1}\mathbf{a}(\varphi_k)$$

By expressing the fact that $\mathbf{w}^H\mathbf{a}(\varphi_k) = 1$, we get $\lambda = (\mathbf{a}^H(\varphi_k)\mathbf{R}^{-1}\mathbf{a}(\varphi_k))^{-1}$. If we replace it in $\mathbf{w}$, we get:

$$\mathbf{w} = \frac{1}{\mathbf{a}^H(\varphi_k)\mathbf{R}^{-1}\mathbf{a}(\varphi_k)}\,\mathbf{R}^{-1}\mathbf{a}(\varphi_k)$$

Replacing this in the expression of the criterion leads us to:

$$\mathbf{w}^H\mathbf{R}\mathbf{w} = \frac{\mathbf{a}^H(\varphi_k)\mathbf{R}^{-1}\mathbf{R}\mathbf{R}^{-1}\mathbf{a}(\varphi_k)}{(\mathbf{a}^H(\varphi_k)\mathbf{R}^{-1}\mathbf{a}(\varphi_k))^2} = \frac{1}{\mathbf{a}^H(\varphi_k)\mathbf{R}^{-1}\mathbf{a}(\varphi_k)}$$

In an estimation problem, the matrix $\mathbf{R}$ is replaced by:

$$\hat{\mathbf{R}}_N = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}(n)\mathbf{x}^H(n)$$

The *Capon method* consists of determining the P maxima of the function:

$$S_{\text{Capon}}(\varphi) = \frac{1}{\mathbf{a}^H(\varphi)\mathbf{R}^{-1}\mathbf{a}(\varphi)} \tag{10.52}$$

This expression should be compared with 10.50.

MUSIC

This approach was presented in detail in paragraph 10.3.3. Just remember that, starting with equation 10.47, we end up with the following expression of the covariance matrix:

$$\mathbf{R} = \mathbf{R}_0 + \sigma^2\mathbf{I}$$

where $\mathbf{R}_0(\varphi) = \mathbf{A}(\varphi)\mathbf{R}_s\mathbf{A}^H(\varphi)$ is assumed to be a matrix with rank P. We showed with equation 10.37 that:

$$\mathbf{R} = \mathbf{V}\Lambda\mathbf{V}^H + \sigma^2\mathbf{G}\mathbf{G}^H$$

where $\mathbf{G}\mathbf{G}^H$ refers to the orthogonal projector onto the noise subspace. The MUSIC algorithm then consists of finding the P maxima of the function:

$$S_{\text{MUSIC}}(\varphi) = \frac{1}{\mathbf{a}^H(\varphi)\mathbf{G}\mathbf{G}^H\mathbf{a}(\varphi)} \tag{10.53}$$

ESPRIT

The ESPRIT algorithm, short for Estimation of Signal Parameters via Rotational Invariant Techniques, was first suggested in [79]. It uses the fact that the antenna can be decomposed in two identical sub-antennas. This approach is typically well-suited for ULA. It also assumes that $\mathbf{R}_s$ is a full rank matrix, and therefore that there is (see property 10.2) a full rank $(P \times P)$ matrix $\mathbf{T}$ such that:

$$\mathbf{V} = \mathbf{A}(\varphi)\mathbf{T}$$

Let $\mathbf{A}_1 = [\mathbf{I}_{M-1} \;\; 0]\mathbf{A}$ and $\mathbf{A}_2 = [0 \;\; \mathbf{I}_{M-1}]\mathbf{A}$. In other words, $\mathbf{A}_1$ represents the first $(M-1)$ lines of $\mathbf{A}$ and $\mathbf{A}_2$ the last $(M-1)$. The expression of $\mathbf{A}(\varphi)$ implies that $\mathbf{A}_2 = \mathbf{A}_1\Omega$ where $\Omega = \text{diag}(e^{j\theta_1}, \ldots, e^{j\theta_P})$ (see equation 10.45).

Let $\mathbf{V}_1 = [\mathbf{I}_{M-1} \;\; 0]\mathbf{V}$ and $\mathbf{V}_2 = [0 \;\; \mathbf{I}_{M-1}]\mathbf{V}$. Because of the expression $\mathbf{V} = \mathbf{A}\mathbf{T}$ $\mathbf{V}_1 = \mathbf{A}_1\mathbf{T}$ and $\mathbf{V}_2 = \mathbf{A}_2\mathbf{T}$, and therefore:

$$\mathbf{V}_2 = \mathbf{V}_1\mathbf{T}^{-1}\Omega\mathbf{T}$$

The pseudo-inverse of $\mathbf{V}_1$ is denoted by $\mathbf{V}_1^{\#} = (\mathbf{V}_1^H\mathbf{V}_1)^{-1}\mathbf{V}_1^H$. By definition $\mathbf{V}_1^{\#}\mathbf{V}_1 = \mathbf{I}$. This means that we can write:

$$\mathbf{T}\mathbf{V}_1^{\#}\mathbf{V}_2 = \Omega\mathbf{T}$$

Notice that $\mathbf{V}_1^{\#}\mathbf{V}_2$ is a $(P \times P)$ matrix.

We are now going to show that, if $\mathbf{TA} = \mathbf{BT}$, then $\mathbf{A}$ and $\mathbf{B}$ have the same eigenvalues. This is because if λ is an eigenvalue of $\mathbf{A}$ associated with the eigenvector $\mathbf{u}$, meaning that $\mathbf{Au} = \lambda\mathbf{u}$, then we have $\mathbf{TAu} = \lambda\mathbf{Tu}$ on one hand, and $\mathbf{TAu} = \mathbf{BTu}$ on the other, and therefore $\mathbf{B} \times \mathbf{Tu} = \lambda \times \mathbf{Tu}$ which leads to the fact that λ is an eigenvalue of $\mathbf{B}$ associated with the eigenvector $\mathbf{Tu}$. Because of this property, and because Ω is a diagonal matrix, we have the following result.

Property 10.4 (ESPRIT estimation) *The ESPRIT estimator of the parameter $\boldsymbol{\theta}$ associated with the model 10.47 (see also 10.33) is $\hat{\boldsymbol{\theta}}_{\text{ESPRIT}} = [\theta_1, \ldots, \theta_P]$ where the $e^{j\theta_1}, \ldots, e^{j\theta_P}$ are the P eigenvalues of the matrix $\mathbf{V}_1^{\#}\mathbf{V}_2$ where $\mathbf{V}_1 = [\mathbf{I}_{M-1} \;\; 0]\mathbf{V}$, $\mathbf{V}_2 = [0 \;\; \mathbf{I}_{M-1}]\mathbf{V}$ and where $\mathbf{V}$ is defined by equation 10.37.*

The ESPRIT algorithm uses this property: the covariance matrix is estimated based on a sequence of observations $\mathbf{x}(n)$, and its eigendecomposition leads to the matrix $\mathbf{V}$ associated with the signal subspace of the P highest eigenvalues. We then have to determine the eigenvalues of $\mathbf{V}_1^{\#}\mathbf{V}_2$. One of the strong points of ESPRIT compared to MUSIC is that it does not require the search for maxima.

Example 10.5 (Comparison of the MUSIC and ESPRIT algorithms)

Simulating three sources and an antenna with eight sensors, we compare the MUSIC and ESPRIT algorithms.

1. Write a MATLAB® function that implements the ESPRIT algorithm.

2. Write a function that implements the MUSIC algorithm for antenna processing:

   – either with the FFT, by computing the function $S_{\text{MUSIC}}(\theta)$ given by expression 10.53 then by determining its P maxima;

   – or by finding the P roots closest to the unit circle with a modulus smaller than 1 of the polynomial $Q(z)$ given by expression 10.43.

3. Write a program that simulates the three sources coming in from the three directions $\varphi_1 = -30°$, $\varphi_2 = 15°$ and $\varphi_3 = 20°$, on an antenna comprising $M = 8$ sensors. The distance between two sensors will be set equal to half the wavelength.

   Evaluate for 100 trials the square deviations for the estimations as functions of the signal-to-noise ratio for $T = 20$ snapshots.

   By referring to condition 10.49, notice that the difference $e = \varphi_3 - \varphi_1$ is smaller than 1/10th of a radian and that therefore $Med/\lambda_0 = 0.4$.

HINT:

1. This function implements the ESPRIT algorithm. Type:

```
function z=esprit_doa(xm,P)
%%=============================================%
%% ESPRIT for DOA                              %
%% SYNOPSIS:  z=ESPRIT_DOA(xm,P)               %
%% xm : (M x N) Observations of                %
%%      N Snapshots on M Sensors (complex data) %
%%  P : Number of Sources (P<M)                %
%%  z : Eigenvalues of pinv(V1)*V2             %
%%=============================================%
[M, N]=size(xm);
xm=xm-(xm*ones(N,N)/N); Rx=zeros(M,M);
for ii=1:N, Rx=Rx+xm(:,ii)*xm(:,ii)'; end
Rx=Rx/N;
[UU dd VV]=svd(Rx);
S1=UU(1:M-1,1:P); S2=UU(2:M,1:P);
PHI=S1 \ S2; vp=eig(PHI); z=-angle(vp)';
return
```

2. This function implements the MUSIC algorithm. Type:

```
function [z]=music_doa(xm,P,method)
%%======================================================%
%% FFT-MUSIC and ROOTS-MUSIC for DOA                    %
%% SYNOPSIS: [z]=MUSIC_DOA(xm,P,method)                 %
%%    xm     = (M x N) Observations of                  %
%%             N Snapshots on M Sensors (complex data) %
%%    P      = Number of Sources (P<M)                  %
%%    method = 'FFT' ou 'ROOTS'                         %
%%    z      = angles of the roots                      %
%%======================================================%
[M, N]=size(xm);
xm=xm-(xm*ones(N,N)/N); Rx=zeros(M,M);
for ii=1:N, Rx=Rx+xm(:,ii)*xm(:,ii)'; end
Rx=Rx/N; [UU dd VV]=svd(Rx); GG=VV(:,P+1:M);
if strncmp(method,'ROOTS',3)
    %===== FFT ROOTS
    PP=GG*GG';
    QQ=zeros(2*M-1,1);
    for d=0:M-1
        for k=0:d, QQ(d+1) = QQ(d+1) + PP(M-d+k,k+1); end
    end
    QQ(M+1:2*M-1)=conj(QQ(M-1:-1:1));
    zz=roots(QQ(2*M-1:-1:1));
    %===== Keep the closest complex roots to
    %      the unit circle
    v=find(abs(zz)<=1 & imag(zz)~=0);
    rac=sort(zz(v)); mr=length(rac);
     rac=rac(mr-P+1:mr);
    z=-angle(rac)';
elseif method=='FFT'
```

```
    %===== FFT MUSIC
    Lfft=1024; GGf=abs(fft(GG,Lfft)) .^2;
    weights=ones(M-P,1); Smusic=1 ./ (GGf*weights);
    diffSmusic=diff(Smusic);
    S0=max(Smusic);
    %===== Look for P maxima
    uM=zeros(1,P);
    for pp=1:P
        [Smax, indmax]=max(Smusic);
        uM(pp)=(indmax-1)/Lfft;
        if uM(pp)>1/2, uM(pp)=-1+uM(pp); end
        %===== Suppress the found maximum
        kkmin=max([indmax-2 1]);
        while diffSmusic(kkmin)>0&kkmin>1,
            kkmin=kkmin-1;
        end
        kkmax=min([indmax Lfft-1]);
        while diffSmusic(kkmax)<0&kkmax<Lfft-1,
            kkmax=kkmax+1;
        end
        Smusic(kkmin:kkmax)=zeros(kkmax-kkmin+1,1);
    end
    z=-uM*2*pi;
else
    error('Method must be: ROOTS or FFT')
end
```

3. Type the following program:

```
%===== TESTESPRITMUSIC.M
% RMS error for DOA Estimation as a function of the SNR
% (ESPRIT and MUSIC methods)
clear
%===== Distance between Sensors/Wavelength
d_on_lambda0 = 1/2;
M = 8;                      % Number of Sensors
%===== Try two sequences of vphi
vphi = sort([-30 10 20]);   % true DOA
%vphi = sort([-30 15 20]);  % true DOA
P = length(vphi);           % Number of Sources
T = 20;                     % Number of Snapshots
%===== SNR list in dB
LISTEsnr=[10:2:18];
ln=length(LISTEsnr);
nbruns=100; %===== Number of Runs (> 2)
eqmE=zeros(ln,P); eqmM=zeros(ln,P);
for bb=1:ln
    snr=LISTEsnr(bb);
    for rr=1:nbruns
        x=Fgene(d_on_lambda0,M,vphi,snr,T);
        zzE=esprit_doa(x,P);
        sinthetaE=zzE/(2*pi*d_on_lambda0);
        ind=find(sinthetaE>1);
        sinthetaE(ind)=ones(length(ind),1);
        ind=find(sinthetaE<-1);
        sinthetaE(ind)=-ones(length(ind),1);
        deltathetaE=...
            sort(asin(sinthetaE)*180/pi)-vphi;
        eqmE(bb,:)=...
            eqmE(bb,:)+deltathetaE.*deltathetaE;
        %===== Try the ROOTS or FFT methods
        zzM=music_doa(x,P,'FFT');
        sinthetaM=zzM/(2*pi*d_on_lambda0);
        ind=find(sinthetaM>1);
        sinthetaM(ind)=ones(length(ind),1);
        ind=find(sinthetaM<-1);
        sinthetaM(ind)=-ones(length(ind),1);
        deltathetaM=...
            sort(asin(sinthetaM)*180/pi)-vphi;
        eqmM(bb,:)=...
            eqmM(bb,:)+deltathetaM.*deltathetaM;
    end
    eqmE(bb,:)=eqmE(bb,:)/nbruns;
    eqmM(bb,:)=eqmM(bb,:)/nbruns;
end
for pp=1:P
    subplot(P,1,pp); subplot(3,1,pp);
    plot(LISTEsnr,eqmE(:,pp),'b');
    subplot(3,1,pp); hold on;
    plot(LISTEsnr,eqmM(:,pp),'r'); hold off
end
```

which uses the signal generating function:

```
function x=Fgene(d_on_lambda0,M,vphi,snr,N)
%%==================================================%
%% Generate N Snapshots on M Sensors for DOA vphi   %
%% SYNOPSIS: x=FGENE(d_on_lambda0,M,vphi,snr,N,OPT) %
%%    d_on_lambda0 = Sensors distance               %
%%                   and wavelength ratio           %
%%    M     = Number of Sensors                     %
%%    vphi = DOA of Sources (P=length(vphi)<M)      %
%%    snr   = Signal to Noise Ratio in dB           %
%%    N     = Number of Snapshots                   %
%%==================================================%
P=length(vphi);
lz=-2*j*pi*(0:M-1)' * sin(vphi*pi/180) * d_on_lambda0 ;
Az=exp(lz);
%===== P Random Sources
```

```
so=randn(P,N); Pso=sum(trace(so*so'))/N;
so=so/sqrt(Pso); sr=zeros(M,N);
for tt=1:N, sr(:,tt)=Az*so(:,tt); end
nvb=sqrt(1/2)*10^(-snr/20);
x=sr+nvb*(randn(M,N)+j*randn(M,N));
return
```

■

Exercise 10.3 (MUSIC 2D) We consider an antenna comprising M sensors assumed to be identical. The vectors $\mathbf{r}_m = [x_m \; y_m \; z_m]^T$, with $m \in \{1, \ldots, M\}$, describe the 3D location of the m-th sensor. $K < M$ sources are assumed to be narrow band with a wavelength λ_0. When the plane wave hypothesis is valid (the sources are far from the antenna), the response of the m-th sensor to the k-th source is given by:

$$a_m(\zeta_k, \varphi_k) = \exp\left(-j\,\mathbf{r}_m^T\,\boldsymbol{\beta}(\zeta_k, \varphi_k)\right)$$

where the wave-vector:

$$\boldsymbol{\beta}_k(\zeta_k, \varphi_k) = \frac{2\pi}{\lambda_0}\,[\sin(\zeta_k)\cos(\varphi_k) \quad \sin(\zeta_k)\sin(\varphi_k) \quad \cos(\zeta_k)]^T$$

and where ζ_k is the elevation and φ_k the azimuth (see Figure 10.9) of the direction of propagation of the k-th source.

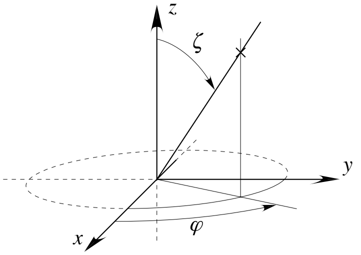


Figure 10.9 – *Angular references*

Under the narrow band assumption, the observed signal may be written:

$$\mathbf{x}(n) = [\mathbf{a}(\zeta_1, \varphi_1) \;\ldots\; \mathbf{a}(\zeta_K, \varphi_K)]\,\mathbf{s}(n) + \mathbf{b}(n)$$

1. Determine the expression of the MUSIC function (see formula 10.53).
2. Write a program:
   - that simulates $N = 100$ samples of $\mathbf{x}(n)$ for $K = 3$ with $\zeta = [30^\circ\, 40^\circ\, 50^\circ]$, $\varphi = [60^\circ\, 50^\circ\, 20^\circ]$ and with an antenna comprising $M = 25$ sensors located on a grid in the plane xOy;
   - that displays the MUSIC function in 2D.

Chapter 11

The Least Squares Method

In this chapter, we are going to present a series of techniques based on minimizing mean square criteria to solve linear problems. But first we are going to state a fundamental theorem, called the *projection theorem.* It was mentioned more or less explicitly in the affine trend suppression problem, or when we estimated the amplitudes of a harmonic signal's components. We will see that it has major applications both in a deterministic or random context.

11.1 The projection theorem

The projection theorem is presented in mathematical form. However, readers that are not used to this formalism should not be worried, since the result expressed by relation 11.1 is quite intuitive, as it is shown in Figure 11.1.

Definition 11.1 (Hilbert space) *Let $\mathcal{H}$ be a vector space with a dot product* $(\mathbf{x}, \mathbf{y})$ *for any two of its elements:*

- *The norm of an element x of $\mathcal{H}$ is the positive number defined by $||\mathbf{x}|| = \sqrt{(\mathbf{x}, \mathbf{x})}$.*
- $\mathbf{x}$ *and* $\mathbf{y}$ *are said to be orthogonal, which is denoted by* $\mathbf{x} \perp \mathbf{y}$*, if* $(\mathbf{x}, \mathbf{y}) = 0$.
- *The distance between two elements* $\mathbf{x}$ *and* $\mathbf{y}$ *of $\mathcal{H}$ is the positive number defined by $d(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}|| = \sqrt{(\mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y})}$.*

$\mathcal{H}$ is said to be a Hilbert space if it is a complete metric space, that is if any Cauchy sequence converges in $\mathcal{H}$.

The following examples are fundamental to signal processing applications:

- **Deterministic case**: the space $\ell^2(\mathbb{Z})$ of square summable complex sequences, that is the sequences such that $\sum_k |x_k|^2 < +\infty$, with the scalar product:

$$(\mathbf{x},\mathbf{y}) = \sum_k x_k^* y_k$$

is a Hilbert space. Finite sequences of length N are an example of a Hilbert space. The set of these sequences can also be seen as the space $\mathbb{C}^N$ of vectors with N complex components.

- **Deterministic case**: the space $L^2(0,T)$ of square summable complex functions, that is the functions such that $\int_0^T |x(t)|^2 dt < +\infty$, with the scalar product:

  $$(\mathbf{x},\mathbf{y}) = \int_0^T x^*(t)y(t)dt$$

  is a Hilbert space.

- **Random case**: the space $L^2(\Omega,\mathcal{F},\mathbb{P})$ comprising the zero-mean random variables defined on a same probability space $(\Omega,\mathcal{F},\mathbb{P})$, and square summable, that is such that $\mathbb{E}\left\{|x|^2\right\} < +\infty$, with the scalar product:

  $$(x,y) = \mathbb{E}\{x^* y\}$$

  is a Hilbert space.

Theorem 11.1 (Projection theorem) *Let $\mathcal{H}$ be a Hilbert space, $\mathcal{C}$ a subspace of $\mathcal{H}$, and let $\mathbf{y}$ be any element of $\mathcal{H}$:*

- *There exists a single element $\mathbf{s}_0 \in \mathcal{C}$ such that:*

  $$\forall \mathbf{s} \in \mathcal{C}, \qquad \|\mathbf{y}-\mathbf{s}_0\|^2 \le \|\mathbf{y}-\mathbf{s}\|^2 \Longleftrightarrow d(\mathbf{y},\mathbf{s}_0) \le d(\mathbf{y},\mathbf{s})$$

- *This element $\mathbf{s}_0$ verifies:*

  $$\forall \mathbf{s} \in \mathcal{C}, \qquad \mathbf{y}-\mathbf{s}_0 \perp \mathbf{s} \tag{11.1}$$

- *The minimal difference has the expression:*

  $$\epsilon^2 = (\mathbf{y}-\mathbf{s}_0, \mathbf{y}-\mathbf{s}_0) = (\mathbf{y}-\mathbf{s}_0, \mathbf{y}) \tag{11.2}$$

Relation 11.1, also sometimes called the *orthogonality principle*, is what is used in practice for determining $\mathbf{s}_0$. $\mathbf{s}_0$ is called the *orthogonal projection* of $\mathbf{y}$ onto $\mathcal{C}$. Figure 11.1 illustrates this property.

We will now examine the applications of the projection theorem for the previously described deterministic and random signals.

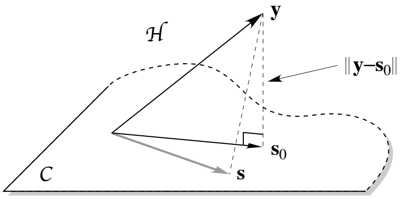


Figure 11.1 – *The projection theorem*

– **Deterministic case**: $\mathcal{H}$ is the space $\mathbb{C}^N$ of the length N vectors with complex components, with the scalar product:

$$(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{N} x_k^* y_k$$

associated with the norm:

$$\|\mathbf{x}\| = \sqrt{\sum_{k=1}^{N} |x_k|^2}$$

Let us choose for $\mathcal{C}$ the sub-space generated by the linear combination of P vectors $\mathbf{x}^1$, $\mathbf{x}^2$, ..., $\mathbf{x}^P$ of $\mathcal{H}$, with $P < N$. Let $\mathbf{X} = [\mathbf{x}^1 \quad \mathbf{x}^2 \quad \dots \quad \mathbf{x}^P]$ be the $(N \times P)$ matrix constructed from the N components of its P vectors. Any vector of $\mathcal{C}$ can then be written:

$$\mathbf{s} = \sum_{k=1}^{P} h_k \mathbf{x}^k = \mathbf{X}\mathbf{h}$$

where $\mathbf{h}$ is a vector with P complex components.

Let $\mathbf{y}$ be another vector of $\mathcal{H}$. The closest element $\mathbf{s}_0$ to $\mathbf{y}$ that belongs to $\mathcal{C}$ is such that $\mathbf{y} - \mathbf{s}_0$ is orthogonal to the P vectors that generate $\mathcal{C}$. Because $\mathbf{s}_0$ belongs to $\mathcal{C}$, it can be written $\mathbf{X}\mathbf{h}$. In order to determine it, we need only express the fact that $\mathbf{y}-\mathbf{X}\mathbf{h}$ is orthogonal to the P columns of $\mathbf{X}$. In matrix form:

$$\mathbf{X}^H(\mathbf{y} - \mathbf{X}\mathbf{h}) = 0 \quad \Longleftrightarrow \quad \mathbf{X}^H\mathbf{X}\mathbf{h} = \mathbf{X}^H\mathbf{y}$$

Knowing $\mathbf{y}$ and $\mathbf{X}$ leads to the coefficient vector $\mathbf{h}$ that allows us to find the projection $\mathbf{s}_0$ we are trying to determine. For example, if $\mathbf{X}^H\mathbf{X}$ is invertible we get:

$$\mathbf{s}_0 = \mathbf{X}(\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H\mathbf{y} \tag{11.3}$$

This result can be applied to the resolution of linear systems of equations, or to the parametric approximation of a sequence of values (exercise 11.1 and example 11.1).

– **Random case**: $\mathcal{H}$ is the space of square summable zero-mean random variables, defined on the same probability space, with the scalar product:

$$(x, y) = \mathbb{E}\{x^* y\}$$

associated with the norm:

$$||x|| = \sqrt{\mathbb{E}\{|x|^2\}}$$

Let us assume that $\mathcal{C}$ is the sub-space generated by the linear combinations of P random variables $x_1, \ldots, x_P$ de $\mathcal{H}$, with $P < N$. Any random variable of $\mathcal{C}$ can then be written:

$$s = \sum_{k=1}^{P} h_k x_k$$

Let y be another random variable of $\mathcal{H}$. s_0, the element closest to y belonging to $\mathcal{C}$, is such that $(y - s_0)$ is orthogonal to the P random variables that generate $\mathcal{C}$. Therefore we are trying to determine s such that, for any $n \in \{1, \ldots, P\}$:

$$\mathbb{E}\{(y - s)x_n^*\} = \mathbb{E}\left\{\left(y - \sum_{k=1}^{P} h_k x_k\right) x_n^*\right\} = 0$$

$$\iff \sum_{k=1}^{P} h_k \mathbb{E}\{x_k x_n^*\} = \mathbb{E}\{y x_n^*\} \tag{11.4}$$

Relation 11.4 then makes it possible to determine the coefficients of h_k by knowing the values of $\mathbb{E}\{y x_n^*\}$ and $\mathbb{E}\{x_k x_n^*\}$. We saw, on page 407, an application of this result to the linear prediction problem. In this case, y is the value of the process at the time n, and the x_j are the values of that same process at the previous P times. We will see another application of this result on page 417 when we discuss the Wiener filter.

COMMENT: the fact that the norm $\mathcal{H}$ is associated with a scalar product plays a fundamental role in the demonstration of the theorem. If we choose other norms, such as the norm $||.||_p$ in the case of the space $\mathbb{C}^N$:

$$||x||_p = \left(\sum_{j=1}^{N} |x_j|^p\right)^{1/p}$$

which is not associated with a scalar product for $p \neq 2$, the theorem does not apply. This is why, independently from its physical significance, which is not always relevant, it is often preferable to use a quadratic approximation criterion since the projection theorem provides a simple analytical solution to the optimization problem.

11.2 The least squares method

C. F. Gauss came up with the idea of the “least squares” method and used it to study the movement of the planets[1]. Based on a sequence of a planet's N positions $(x_1(n), x_2(n))$, $n \in [1..N]$ in its orbit plane (Figure 11.2), the problem was to estimate the parameters characterizing the general equation of an ellipse: $ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 - 1 = 0$.

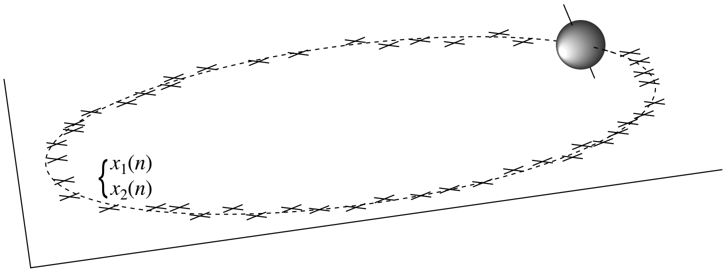


Figure 11.2 – *Originally, the least squares method was created to study the movement of the planets*

Because of measurement errors, the observed values did not all perfectly belong to a single ellipse. Gauss's idea was to choose the values of the parameters that would characterize a “mean” ellipse. Mathematically speaking, the goal is to minimize the sum of the square deviations of the points observed on the ellipse to be determined, which amounts to choosing the values of a, b c, d and e that minimize:

$$\sum_{n=1}^{N} \left(ax_1^2(n) + bx_2^2(n) + cx_1(n)x_2(n) + dx_1(n) + ex_2(n) - 1\right)^2$$

This is actually a rather optimistic objective, since it means we have to assume that if the model works, we wouldn't be wrong, “on average”. As for the solution, it is of a simple form, which is not surprising since setting the derivatives with respect to the various unknowns to zero leads to first degree equations.

11.2.1 Formulating the problem

Consider a sequence of scalar observations $y(n)$, presented as the sum of a signal $s(n; \mathbf{h}_0)$ the type of which is known and of an additive noise $w(n)$. The

[1] In 1801, Gauss calculated the orbit of the asteroid Ceres by observing it for 41 days. At the moment where Ceres became hidden from view because of the Sun's light, Gauss managed to predict where the asteroid would reappear.

vectorial parameter $\mathbf{h}_0$ represents the parameter we wish to estimate. We have:

$$y(n) = s(n;\mathbf{h}_0) + w(n)$$

The noise (n) encompasses both the *measurement noise* and the *modeling noise*, the latter expressing a lack of *a priori* knowledge of the theoretical model $s(n;\mathbf{h}_0)$.

Based on the N observations $\{y(1), \ldots, y(N)\}$, the *least squares estimator* is the value of $\mathbf{h}$ that minimizes the square deviation:

$$J(\mathbf{h}) = \sum_{n=1}^{N} |y(n) - s(n;\mathbf{h})|^2 = (\mathbf{y} - \mathbf{s}(\mathbf{h}_0))^H(\mathbf{y} - \mathbf{s}(\mathbf{h}_0)) \tag{11.5}$$

where $\mathbf{y} = [y(1), \ldots, y(N)]^T$ and $\mathbf{s}(\mathbf{h}_0) = [s(1;\mathbf{h}_0), \ldots, s(N;\mathbf{h}_0)]^T$. We already encountered this type of problem in paragraph 10.1. The goal was to estimate the frequency and the amplitude of the complex exponential $s(n;\mathbf{h}_0) = \alpha^0 e^{2j\pi f_1^0 n}$ corrupted by noise. In that case, the least squares method consisted of determining $\mathbf{h} = (f_1, \alpha)$ such that:

$$\sum_{n=1}^{N} \left| y(n) - \alpha e^{2j\pi f_1 n} \right|^2$$

would be minimum. We saw that the solution suggested in paragraph 10.1 did not have a simple analytical form for f_1 because the expression of $s(n;\mathbf{h})$ is not linear with respect to f_1. However, when the model is linear, the problem can easily be solved.

11.2.2 The linear model

Let us now assume that the signal $s(n;\mathbf{h})$ is expressed linearly as a function of the unknown parameter $\mathbf{h}_0 = [h(0) \quad \cdots \quad h(P-1)]^T$ according to the expression:

$$s(n;\mathbf{h}_0) = x_{n,1}h(0) + \cdots + x_{n,P}h(P-1)$$

The known quantities $x_{n,j}$ are sometimes referred to as *regressors*. Therefore the observation has the expression $y(n) = x_{n,1}h(0) + \cdots + x_{n,P}h(P-1) + w(n)$ where $w(n)$ refers to the noise. By stacking N successive observations, we get the matrix expression:

$$\begin{bmatrix} y(1) \\ \vdots \\ y(N) \end{bmatrix} = \begin{bmatrix} x_{1,1} & \ldots & x_{1,P} \\ \vdots & & \vdots \\ x_{N,1} & \ldots & x_{N,P} \end{bmatrix} \begin{bmatrix} h(0) \\ \vdots \\ h(P-1) \end{bmatrix} + \begin{bmatrix} w(1) \\ \vdots \\ w(N) \end{bmatrix}$$

which can be written, using obvious notations:

$$\mathbf{y} = \mathbf{X}\mathbf{h}_0 + \mathbf{w} \tag{11.6}$$

where $\mathbf{h}_0 = [h(0) \quad \cdots \quad h(P-1)]^T$ refers to the actual value and where $\mathbf{X}$ is the $(N \times P)$ matrix with $x_{n,j}$ as its generating element.

11.2.3 The least squares estimator

Starting with N observations $\mathbf{y} = [y(1), \ldots, y(N)]^T$, the *least squares estimator* is the argument $\mathbf{h}$ that minimizes the square deviation:

$$J(\mathbf{h}) = (\mathbf{y} - \mathbf{Xh})^H(\mathbf{y} - \mathbf{Xh}) \tag{11.7}$$

First consider the case of a noiseless observation expressed as $\mathbf{y} = \mathbf{Xh}_0$. To estimate $\mathbf{h}_0$, we need as many equations as there are unknowns, hence we choose $N = P$. If we assume that the square matrix $\mathbf{X}$ is invertible, the solution can be written simply as:

$$\mathbf{h} = \mathbf{X}^{-1}\mathbf{y}$$

Notice that in this case, the square deviation, expression 11.7, is precisely equal to 0. Therefore, in the absence of noise, the observation of P of the signal's values is enough to estimate $\mathbf{h}_0$, and to do so without making an error.

In the presence of noise, this is no longer the case: there usually is no vector $\mathbf{h}$ that simultaneously verifies the N equations $\mathbf{y} = \mathbf{Xh}$, that is to say such that $J(\mathbf{h}) = 0$. At best, we can hope to find a vector $\mathbf{h}$ that minimizes $J(\mathbf{h})$. But then why would we choose $N \gg P$? The answer is given by property 11.1 on page 397, which states that the least squares estimator's variance usually decreases as N increases. Although the least squares method was not originally meant for probabilistic purposes, this result justifies its use in the presence of additive noise.

We now go back to the minimization problem, to consider the case where the number N of equations is greater than the number P of unknowns: the system is said to be *over-determined*. The fact that there is no vector $\mathbf{h}$ such that $\mathbf{Xh}$ is exactly equal to the observation vector $\mathbf{y}$ is equivalent to saying that the length N vector $\mathbf{y}$ does not belong to the vector subspace generated by the P column vectors of $\mathbf{X}$. This vector subspace is called the *image* of $\mathbf{X}$ and is denoted by Image($\mathbf{X}$). The least squares method consists of finding the vector of Image($\mathbf{X}$) closest to $\mathbf{y}$ in terms of square distance, hence the solution is given by the *projection theorem.*

Solving

Using the same notations as in the projection theorem, $\mathcal{H}$ is the space $\mathbb{C}^N$ of length N vectors and $\mathcal{C} = \text{Image}(\mathbf{X})$ is the subspace generated by the P column vectors of $\mathbf{X}$. Any vector of $\mathcal{C}$ can be written $\mathbf{Xh}$ where $\mathbf{h}$ is any length P vector. The orthogonality principle states that the vector $\mathbf{Xh} \in \mathcal{C}$ we wish to determine is such that $(\mathbf{y} - \mathbf{Xh})$ is orthogonal to the columns of $\mathbf{X}$. This can be written in matrix form as follows:

$$\mathbf{X}^H(\mathbf{y} - \mathbf{Xh}) = 0 \iff \mathbf{X}^H\mathbf{Xh} = \mathbf{X}^H\mathbf{y} \tag{11.8}$$

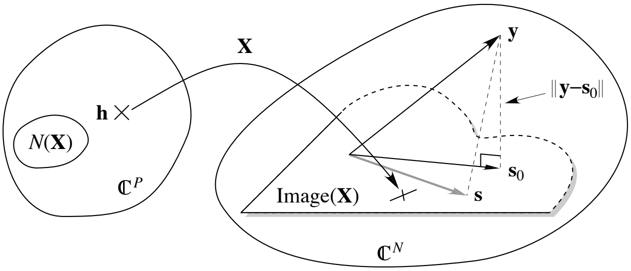


Figure 11.3 – *Image of* X

Figure 11.3 shows a representation of Image($\mathbf{X}$), as well as of the orthogonal projection of $\mathbf{y}$ onto Image($\mathbf{X}$).

Remember that the orthogonal projection of y onto $\mathcal{C}$ is *unique*. This does not mean, however, that $\mathbf{h}$ is unique. We have the following results:

– If the rank of $\mathbf{X}$ is P, in other words if $\mathbf{X}$ is a *full rank matrix*, then the P column vectors of $\mathbf{X}$ are independent, and the $(P \times P)$ matrix $\mathbf{X}^H\mathbf{X}$ is invertible, and the solution we are trying to determine has the expression:

$$\mathbf{h} = (\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H\mathbf{y} \tag{11.9}$$

In the particular case where $P = N$, the matrix $\mathbf{X}$ is square, and $(\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H = \mathbf{X}^{-1}$. We end up with the usual inverse of a linear system of P equations with P unknowns.

– If the rank of $\mathbf{X}$ is such that $R < P$, there are $P - R$ independent length P vectors, referred to as $\mathbf{u}$, such that $\mathbf{Xu} = 0$. The space $N(\mathbf{X})$, the dimension of which is generated by these vectors, is called the *kernel* of $\mathbf{X}$.

In that case, the orthogonal projection is always unique. However, the vector $\mathbf{h}$ is no longer unique: let $\mathbf{h}$ be a vector that verifies 11.8, that is to say such that $\mathbf{X}^H\mathbf{Xh} = \mathbf{X}^H\mathbf{y}$. Then for any $\mathbf{u} \in N(\mathbf{X})$, $\mathbf{X}^H\mathbf{X}(\mathbf{h} + \mathbf{u}) = \mathbf{X}^H\mathbf{Xh} + \mathbf{0} = \mathbf{X}^H\mathbf{y}$ and hence the vector $\mathbf{g} = \mathbf{h} + \mathbf{u}$ also verifies the equation $\mathbf{X}^H\mathbf{Xg} = \mathbf{X}^H\mathbf{y}$. Therefore, there is an infinite number of solutions to equation 11.8. They are all completely defined, except for an additive vector belonging to the kernel of $\mathbf{X}$. Among all these vectors, one of them has the minimum norm, the one orthogonal to the kernel of $\mathbf{X}$. In terms of square deviation values, the solutions are of course all equivalent.

The conclusion is that there is always at least one solution to the equation $\mathbf{X}^H\mathbf{Xh} = \mathbf{X}^H\mathbf{y}$, called the *least squares estimator*, and we will denote it with:

$$\mathbf{h} = \mathbf{X}^{\#}\mathbf{y} \tag{11.10}$$

$\mathbf{X}^{\#}$ is called the *pseudo-inverse* of $\mathbf{X}$. There are two ways of calculating the pseudo-inverse with MATLAB®:

- The function `pinv`, which is used as follows:

```
h=pinv(X)*y
```

- The \ operator, which is used as follows:

```
h= X \ y
```

These two computations only provide the same result if $\mathbf{X}$ is a full rank matrix ($R = P$). Otherwise, `pinv` returns among all possible solutions the minimum norm solution, whereas \ returns the solution that has at most R non-zero components. Of course, these two solutions lead to the same square deviation, and without any further constraint, neither is better than the other. If $\mathbf{X}$ is a full rank matrix, we can always use the expression:

$$\mathbf{X}^{\#} = (\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H \tag{11.11}$$

which then leads to $\mathbf{h} = (\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H\mathbf{y}$. However, the computation with MATLAB® can be numerically less accurate than with the other two methods.

Property 11.1 (Variance of the least squares estimator) *Consider the model* $\mathbf{y} = \mathbf{X}\mathbf{h}_0 + \mathbf{w}$ *where* $\mathbf{w}$ *is assumed to be a white zero-mean noise with the covariance* $\sigma^2\mathbf{I}$*.* $\mathbf{X}^H\mathbf{X}$ *is assumed to be invertible, and* $\mathbf{h} = (\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H\mathbf{y}$ *denotes the least squares estimator. Then* $\mathbf{h}$ *is an unbiased estimator of* $\mathbf{h}_0$ *and the cumulated variance of all the components of* $\mathbf{h}$ *is given by:*

$$\sigma_{\mathbf{h}}^2 = \sigma^2 \mathrm{Tr}((\mathbf{X}^H\mathbf{X})^{-1}) \tag{11.12}$$

HINT: by replacing $\mathbf{y} = \mathbf{X}\mathbf{h}_0 + \mathbf{w}$ in $\mathbf{h} = (\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H\mathbf{y}$, we get:

$$\mathbf{h} = \mathbf{h}_0 + (\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H\mathbf{w} \tag{11.13}$$

If we change over to the expectation on both sides and use the fact that $\mathbb{E}\{\mathbf{w}\} = 0$, we have $\mathbb{E}\{\mathbf{h}\} = \mathbf{h}_0$, hence the least squares estimator is unbiased. Using 11.13, we can also write:

$$(\mathbf{h} - \mathbf{h}_0)(\mathbf{h} - \mathbf{h}_0)^H = (\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H\mathbf{w}\mathbf{w}^H\mathbf{X}(\mathbf{X}^H\mathbf{X})^{-1}$$

If we change over to the expectation of the two members, we infer that the covariance matrix of $\mathbf{h}$ defined by:

$$\mathbf{C}_{\mathbf{h}} = \mathbb{E}\left\{(\mathbf{h} - \mathbf{h}_0)(\mathbf{h} - \mathbf{h}_0)^H\right\}$$

has the expression:

$$\mathbf{C}_{\mathbf{h}} = \sigma^2(\mathbf{X}^H\mathbf{X})^{-1}$$

where we have used the fact that $\mathbb{E}\left\{\mathbf{w}\mathbf{w}^H\right\} = \sigma^2\mathbf{I}$. Finally, we get:

$$\sigma^2_{\mathbf{h}} = \mathbb{E}\left\{(\mathbf{h}-\mathbf{h}_0)^H(\mathbf{h}-\mathbf{h}_0)\right\} = \mathrm{Tr}(\mathbf{C}_{\mathbf{h}}) = \sigma^2\mathrm{Tr}((\mathbf{X}^H\mathbf{X})^{-1})$$

Notice that $\sigma^2_{\mathbf{h}}$ is the sum of the variances of the P components of the estimator $\mathbf{h}$. ■

In many practical problems, the sequence of the regressors involved with the matrix $\mathbf{X}$ is chosen so that the matrix $\mathbf{X}^H\mathbf{X}/N$ is almost equal to a matrix of the type $s^2\mathbf{I}_P$ where $\mathbf{I}_P$ is the $(P\times P)$ identity matrix. In that case, $(\mathbf{X}^H\mathbf{X})^{-1} \approx \mathbf{I}_p/Ns^2$ and $\mathrm{Tr}((\mathbf{X}^H\mathbf{X})^{-1}) \approx P/Ns^2$. By replacing this in 11.12, we get:

$$\sigma^2_{\mathbf{h}} \approx \frac{\sigma^2}{s^2}\frac{P}{N}$$

This result shows that the smaller the variance is, the higher the number N of observations, the higher the ratio $\rho = s^2/\sigma^2$ and the smaller the dimension of the parameter we are trying to identify.

Finally, note that expression 11.9 of the least squares estimator has a linear form with respect to the observation $\mathbf{y}$. Furthermore, it can be proven [74] that it is, among all the unbiased linear estimators, the one with the minimum covariance in the case of white noise. It is called the *BLUE* (*Best Linear Unbiased Estimator*).

In the property 11.1, if $\mathbf{w}$ is assumed to be a zero-mean noise with the covariance $\sigma^2\mathbf{C}$, it is easy to prove by whitening with $\mathbf{C}^{-1/2}$ that:

$$\mathbf{h} = (\mathbf{X}^H\mathbf{C}^{-1}\mathbf{X})^{-1}\mathbf{X}^H\mathbf{C}^{-1}\mathbf{y} \tag{11.14}$$

is the best linear unbiased estimator (BLUE) and the covariance matrix of $\mathbf{h}$ defined by 11.14 has the expression:

$$\mathbf{C}_{\mathbf{h}} = \sigma^2(\mathbf{X}^H\mathbf{C}^{-1}\mathbf{X})^{-1}$$

Weighted least squares

Expression 11.9 can be generalized by considering the scalar product defined in $\mathbb{C}^N$ by the expression:

$$\mathbf{y}^H\mathbf{W}\mathbf{y} \tag{11.15}$$

where $\mathbf{W}$ refers to a positive matrix called a *weighting matrix*. $\mathbf{W}$ is typically used to take into account the fact that the measurement noise is not white or that the observations are not quite “stationary”. In this last case, the first components of $\mathbf{y}$ should probably be given less importance. The classic example is to take:

$$\mathbf{W} = \begin{bmatrix} \lambda^{N-1} & 0 & \cdots & 0 \\ 0 & \lambda^{N-2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} \tag{11.16}$$

where the value $\lambda \in (0,1)$ is called the *forget factor.* If λ is close to 0, the "past" values of $\mathbf{y}$ are assigned a very small weight.

If we once more apply the orthogonality principle, but use the scalar product defined by expression 11.15, equation 11.8 can be rewritten as follows:

$$\mathbf{X}^H\mathbf{W}(\mathbf{y} - \mathbf{X}\mathbf{h}) = 0$$

If we assume that $\mathbf{X}^H\mathbf{W}\mathbf{X}$ is invertible, we infer the expression of the *weighted least squares* estimator:

$$\mathbf{h} = (\mathbf{X}^H\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^H\mathbf{W}\mathbf{y} \tag{11.17}$$

This leads to the same expression as 11.9 when choosing $\mathbf{W} = \mathbf{I}_N$.

We are now going to see a few examples that use a least squares estimator. In these examples, we chose $\lambda = 1$. This study can of course be completed by considering a forget factor $\lambda < 1$.

Example 11.1 (Polynomial approximation)
We are going to try to approximate the function $\sin(x)$ by a $(P-1)$ degree polynomial on the range $(-\pi, +\pi)$. Let $h_0, \ldots, h_{P-1}$ be the P coefficients of this polynomial. To achieve the approximation, we have decided to minimize the square deviation between the polynomial we wish to determine and the function $\sin(x)$ for a set of values of x from $-\pi$ to $+\pi$ by steps of Δ. Let $\{x_1, \ldots, x_N\}$ be the set of these values and $y_n = \sin(x_n)$ the corresponding values.

COMMENT: in practice, the pairs of numerical values such as $(x_1, y_1), \ldots, (x_N, y_N)$ are obtained through a series of measures, and we then try to find a polynomial approximation that gives y as a function of x over the complete range that was chosen.

1. By writing down the values s_n of the polynomial we are trying to determine at the points $x_1, \ldots, x_N$, show that the least squares optimization problem can be interpreted as the orthogonal projection of the vector $[y_1, \ldots, y_N]^T$ onto a space generated by P vectors constructed from $x_1, \ldots, x_N$. Give the least squares solution.

2. Write a program that performs the approximation. Check the quality of the results on the $(-\pi, +\pi)$ grid by steps of $\Delta = 0.2$.

Hint:

1. Let $x_1, \ldots, x_N$ be the values used to perform the approximation and $y_1, \ldots, y_N$ the ones corresponding to the sine function. We wish to approximate the y_n by an expression of the type:

   $$s_n = h_0 + h_1 x_n + \cdots + h_{P-1} x_n^{P-1}$$

   If we stack the s_n we get a vector generated by the P column vectors $\mathbf{e}_j = [x_1^j, x_2^j, \ldots, x_n^j]^T$ where $j \in \{0, 1, \ldots, P-1\}$. Hence the problem is equivalent to finding the best approximation for $\mathbf{y} = [y_1, \ldots, y_N]^T$ belonging to the space generated by the $\mathbf{e}_j$. Using the projection theorem, we find that $\mathbf{h}$ verifies $\mathbf{X}^T\mathbf{X}\mathbf{h} = \mathbf{X}^T\mathbf{y}$ where $\mathbf{X}$ is the $(N \times P)$ matrix defined by $\mathbf{X} = [\mathbf{e}_0, \ldots, \mathbf{e}_{P-1}]$. It can be proved that if $x_i \neq x_j$ for any pair (i, j) with $i \neq j$, then $\mathbf{X}$ is a full rank matrix.

2. The following programs returns Figure 11.4:

```
%===== APPROXSIN.M
% Approximation of a function
% with a (p-1) degree polynomial
x=(-pi:.2:pi)'; N=length(x);
y=sin(x);            % Function to be approximated
pm1=14;              % Polynomial degree
X=zeros(N,pm1);
%===== Matrix X
for ii=1:pm1, X(:,ii)=x .^(ii-1); end
%===== Solving the system
h=X \ y;
%===== Verification
x=(-pi:.01:pi)'; N=length(x); y=sin(x); X=zeros(N,pm1);
for ii=1:pm1, X(:,ii)=x .^(ii-1); end
s=X*h; plot(x,y-s); grid
```

■

Exercise 11.1 (Determining the TF using the gain)
We are going to determine the coefficients of the transfer function:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_q z^{-q}}{1 + a_1 z^{-1} + \cdots + a_p z^{-p}}$$

of a filter, that is the $p + q + 1$ unknowns $b_0, \ldots, b_q, a_1, \ldots, a_p$ from the N values of the gain $|H(e^{2j\pi f})|$ taken at the frequency points $f \in \{f_1, \ldots, f_N\}$.

We will assume that $N > p + q + 1$, and that the calculated filter is causal, stable and minimum phase.

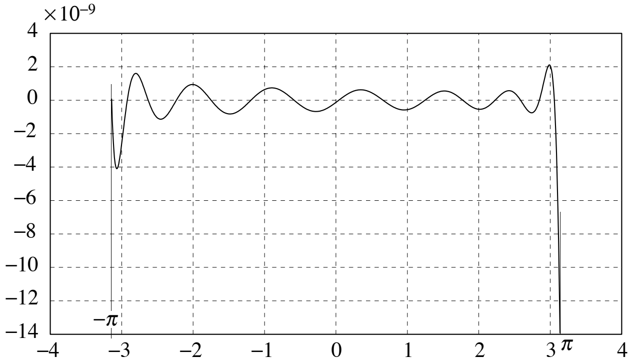


Figure 11.4 – *Error resulting from the polynomial approximation*

1. Show that the problem is equivalent to solving a *linear system* of N equations with $p+q+1$ unknowns.

2. Determine the values of $b_0, \ldots, b_q, a_1, \ldots, a_p$ that lead to the best least squares approximation. Use the result to find the causal, stable, minimum phase filter with the same gain as the obtained filter.

3. Write a MATLAB® function that determines the filter's coefficients from the sequence of the pairs (frequency, gain) and from the degrees of the numerator and denominator.

4. Use this function to create a real low-pass filter with the canceling frequency 0.2 for the frequency sequence `F=(0:0.01:0.5)`. Compare the results with the theoretical graph, by testing different values of p and q.

COMMENT: the method used here does not allow you to determine the best least squares approximation when you set the gain, the phase, and the stability constraint.

Exercise 11.2 (Approximating the inverse of an FIR filter)

1. Consider the FIR filter with the transfer function:

$$H(z) = 1 - 3.4z^{-1} + 1.2 = (1 - 3z^{-1})(1 - 0.4z^{-1})$$

Determine the expression of the impulse response $g(n)$ of the *stable* filter with the transfer function $G(z) = 1/H(z)$. Is this filter causal?

2. We wish to approximate the filter $G(z)$, theoretically IIR and non-causal, by a FIR filter. In order to do this, C coefficients are assigned to the

approximation of the causal part, and A coefficients to the anticausal part. Under these conditions, the impulse response of the filter we are trying to determine is written $g(n) = \{g(-A), \dots, g(0), \dots, g(C)\}$. Of course, when used in real time, the anticausal part will result in a processing delay of A samples.

By writing the convolution of $g(n)$ with $h(n)$, show that the problem can be expressed in a linear way. What sequence of values must be used for approximating the sequence of values $(g \star h)(n)$?

3. Using the least squares method, write a program that calculates $g(-A)$, $\dots$, $g(0)$, $\dots$, $g(C)$ for given values of A and C. Justify the values chosen for A and C.

11.2.4 The RLS algorithm (recursive least squares)

According to expressions 11.9 and 11.14, the calculation of $\mathbf{h}$ requires us to handle an $(N \times P)$ matrix $\mathbf{X}$. If this matrix has to be constructed from a continuous flow of data, and if the calculations have to be performed in *real time*, the limitless increase of the size N makes a direct calculation impossible.

The recursive least squares algorithm brings a solution to this problem. It also has the advantage of not requiring matrix inversion calculations. It works by updating the value of $\mathbf{h}$ as the data pours in.

We are going to start with the expression of the weighted least squares (expression 11.17) and assign the index n to the solution obtained at the n-th step:

$$\mathbf{h}_n = \mathbf{Q}_n \mathbf{X}_n^H \mathbf{W}_n \mathbf{y}_n \tag{11.18}$$

where the matrix $\mathbf{Q}_n = (\mathbf{X}_n^H \mathbf{W}_n \mathbf{X}_n)^{-1}$ is such that $\mathbf{Q}_n = \mathbf{Q}_n^H$. If we choose for $\mathbf{W}_n$ the matrix given by expression 11.16, we have:

$$\mathbf{W}_{n+1} = \begin{bmatrix} \lambda \mathbf{W}_n & \mathbf{0} \\ \mathbf{0}^H & 1 \end{bmatrix}$$

If we denote by $\mathbf{x}_{n+1}^H$ the $(n+1)$-th line of the matrix $\mathbf{X}_{n+1}$, we have:

$$\mathbf{X}_{n+1} = \begin{bmatrix} \mathbf{X}_n \\ \mathbf{x}_{n+1}^H \end{bmatrix}$$

This leads us to:

$$\begin{aligned} \mathbf{X}_{n+1}^H \mathbf{W}_{n+1} \mathbf{X}_{n+1} &= \begin{bmatrix} \mathbf{X}_n^H & \mathbf{x}_{n+1} \end{bmatrix} \mathbf{W}_{n+1} \begin{bmatrix} \mathbf{X}_n \\ \mathbf{x}_{n+1}^H \end{bmatrix} \\ &= \lambda \mathbf{X}_n^H \mathbf{W}_n \mathbf{X}_n + \mathbf{x}_{n+1} \mathbf{x}_{n+1}^H \end{aligned}$$

Hence, the matrix $\mathbf{Q}_n$ verifies the recursive formula:

$$\mathbf{Q}_{n+1} = (\lambda \mathbf{Q}_n^{-1} + \mathbf{x}_{n+1} \mathbf{x}_{n+1}^H)^{-1}$$

We can easily check that if $\mathbf{R}$ is a matrix and $\mathbf{u}$ is a vector of the appropriate length, then:

$$(\mathbf{R}+\mathbf{u}\mathbf{u}^H)^{-1} = \mathbf{R}^{-1} - \frac{\mathbf{R}^{-1}\mathbf{u}\mathbf{u}^H\mathbf{R}^{-1}}{1+\mathbf{u}^H\mathbf{R}^{-1}\mathbf{u}} \tag{11.19}$$

All we need to do is multiply the member on the right by $(\mathbf{R}+\mathbf{u}\mathbf{u}^H)$ to end up with the identity. If we apply the identity 11.19 to $\mathbf{Q}_{n+1}$, we get:

$$\mathbf{Q}_{n+1} = \lambda^{-1}\mathbf{Q}_n - \frac{\lambda^{-2}\mathbf{Q}_n\mathbf{x}_{n+1}\mathbf{x}_{n+1}^H\mathbf{Q}_n}{1+\lambda^{-1}\mathbf{x}_{n+1}^H\mathbf{Q}_n\mathbf{x}_{n+1}}$$

By replacing this expression in that of $\mathbf{h}_{n+1}$, given by 11.18, we have:

$$\begin{aligned}
\mathbf{h}_{n+1} &= \mathbf{Q}_{n+1}\mathbf{X}_{n+1}^H\mathbf{W}_{n+1}\mathbf{y}_{n+1}\\
&= \mathbf{Q}_{n+1}\begin{bmatrix}\mathbf{X}_n^H & \mathbf{x}_{n+1}\end{bmatrix}\begin{bmatrix}\lambda\mathbf{W}_n & \mathbf{0}\\ \mathbf{0}^H & 1\end{bmatrix}\begin{bmatrix}\mathbf{y}_n\\ y_{n+1}\end{bmatrix}\\
&= \mathbf{Q}_{n+1}(\lambda\mathbf{X}_n^H\mathbf{W}_n\mathbf{y}_n + \mathbf{x}_{n+1}y_{n+1})\\
&= \left(\lambda^{-1}\mathbf{Q}_n - \frac{\lambda^{-2}\mathbf{Q}_n\mathbf{x}_{n+1}\mathbf{x}_{n+1}^H\mathbf{Q}_n}{1+\lambda^{-1}\mathbf{x}_{n+1}^H\mathbf{Q}_n\mathbf{x}_{n+1}}\right)(\lambda\mathbf{X}_n^H\mathbf{W}_n\mathbf{y}_n + \mathbf{x}_{n+1}y_{n+1})
\end{aligned}$$

If we develop, using 11.18 and by noticing that $\mathbf{x}_{n+1}^H\mathbf{Q}_n\mathbf{x}_{n+1}$ is a scalar, we get:

$$\mathbf{h}_{n+1} = \mathbf{h}_n + \frac{\lambda^{-1}\mathbf{Q}_n\mathbf{x}_{n+1}}{1+\lambda^{-1}\mathbf{x}_{n+1}^H\mathbf{Q}_n\mathbf{x}_{n+1}}(y_{n+1} - \mathbf{x}_{n+1}^H\mathbf{h}_n)$$

In this expression, the term $(y_{n+1} - \mathbf{x}_{n+1}^H\mathbf{h}_n)$ is expressed as the difference between the observed value y_{n+1} and the value we would have observed if the value $\mathbf{h}_n$ had been exact at the time $(n+1)$. The value $\mathbf{h}_{n+1}$ is obtained by adding to the value $\mathbf{h}_n$ calculated previously a corrective term proportional to this difference. We will see in paragraph 12.14 that the Kalman algorithm has a similar structure.

If we gather the results, the RLS algorithm can be written:

Initial values:
$\mathbf{Q}_0 = \mathbf{I}_P/\delta$ *with* $\delta \ll 1$ *and* $\mathbf{h}_0 = \mathbf{0}$
Repeat for $n = 0 \ldots$*:*

$$\begin{aligned}
\mathbf{K}_{n+1} &= \frac{\lambda^{-1}\mathbf{Q}_n\mathbf{x}_{n+1}}{1+\lambda^{-1}\mathbf{x}_{n+1}^H\mathbf{Q}_n\mathbf{x}_{n+1}}\\
\mathbf{h}_{n+1} &= \mathbf{h}_n + \mathbf{K}_{n+1}(y_{n+1} - \mathbf{x}_{n+1}^H\mathbf{h}_n)\\
\mathbf{Q}_{n+1} &= \lambda^{-1}\mathbf{Q}_n - (1+\lambda^{-1}\mathbf{x}_{n+1}^H\mathbf{Q}_n\mathbf{x}_{n+1})\mathbf{K}_{n+1}\mathbf{K}_{n+1}^H
\end{aligned} \tag{11.20}$$

We could consider starting with the exact initial conditions by calculating $\mathbf{Q}_P = (\mathbf{X}_P^H\mathbf{W}_P\mathbf{X}_P)^{-1}$ and $\mathbf{h}_P = \mathbf{Q}_P\mathbf{X}_P^H\mathbf{W}_P\mathbf{Y}_P$, but if the value of the size N is high, this precaution is not necessary.

Example 11.2 (AR-P estimation by the RLS algorithm)
Consider the recursive equation describing an AR-P process:

$$x(n) + a_1x(n-1) + \cdots + a_Px(n-P) = w(n)$$

where $A(z) = z^P + a_1z^{P-1} + \cdots + a_P$ has all of its roots outside the unit circle and where $w(n)$ is a zero-mean white noise with a variance of σ^2. Show that the problem of estimating the parameters a_1, ..., a_P can be formulated as the expression 11.6 of a linear model. Use this result to estimate the parameters of an AR-P process using the recursive least squares algorithm.

HINT: notice that $x(n) = -a_1x(n-1) - \cdots - a_Px(n-P) + w(n)$. If we let $y(n) = x(n)$, $\mathbf{x}_n = [x(n-1) \quad \cdots \quad x(n-P)]^T$ and $\mathbf{h} = -[a_1 \quad \cdots, a_P]$, we can write $y(n) = \mathbf{x}_n^T\mathbf{h} + w(n)$. By stacking the N successive values, we get an expression of the same type as 11.6. To estimate $\mathbf{h}$, we can therefore use the recursive least squares algorithm. This method is implemented by the following function. Type:

```
function [a]=estARrls(x,P)
%%=============================================%
%% Estimation of the P coefficients of an AR-P %
%% model using the RLS algorihtm               %
%% SYNOPSIS: [a]=ESTARRLS(x,P)                 %
%%     x = Signal                              %
%%     P = Model order                         %
%%     a = [1 a_1 ... a_P]                     %
%%=============================================%
N=length(x); x=x(:); x=x-ones(1,N)*x/N;
hn=zeros(P,1); delta=10^-6; Qn=eye(P)/delta;
%===== (Correlations Method)
x=[zeros(P,1);x;zeros(P,1)];
for k=P+1:N+2*P,
    xn=x(k-1:-1:k-P); yn=x(k);
    Gn=Qn /(1+xn'*Qn*xn); Kn=Gn*xn; en=yn-xn'*hn;
    hn=hn+Kn*en; Qn=Qn-Kn*xn'*Qn;
end
a=[1 ; -hn];
return
```

The correlation method and the covariance method (see page 295) can both be implemented, but the former was chosen, which is why the function returns the same result as the `xtoa` function. The differences noticed are due to the fact that the initial conditions are not the exact conditions. However, it does not require the prior estimation of the covariances. ■

11.2.5 Identifying the impulse response of a channel

Formulating the problem

The objective in a number of problems is to find a model for describing the communication channel as a simple FIR filter. Let $\{h(0), \ldots, h(P-1)\}$ be its impulse response. In order to achieve certain processes, the $h(k)$ have to be estimated. This is called, in this context, channel *identification.*

The easiest method consists of sending a *known* signal, called a *training signal*, and to observe the channel's output. In practice, if the channel varies with time, we have to repeat this operation by periodically sending the training signal. The channel output observation can be written:

$$y(n) = h(0)x(n) + \cdots + h(P-1)x(n-P+1) + w(n)$$

where $w(n)$ represents an additive noise superimposed onto the transmitted signal (Figure 11.5).

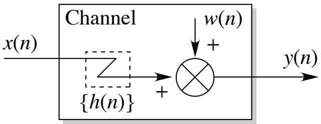


Figure 11.5 – *Channel identification*

If we write the observation sequence in matrix form for n from P to $N+P-1$, we get:

$$\begin{bmatrix} y(P) \\ \vdots \\ y(N+P-1) \end{bmatrix} = \begin{bmatrix} x(P) & \cdots & x(1) \\ \vdots & & \vdots \\ x(N+P-1) & \cdots & x(N) \end{bmatrix} \begin{bmatrix} h(0) \\ \vdots \\ h(P-1) \end{bmatrix} + \begin{bmatrix} w(P) \\ \vdots \\ w(N+P-1) \end{bmatrix} = \mathbf{Xh} + \mathbf{w}$$

With the same matrix notations as before, the vector $\mathbf{h}$ that minimizes the difference $(\mathbf{y} - \mathbf{Xh})^H(\mathbf{y} - \mathbf{Xh})$ is given by expression 11.9 rewritten below:

$$\mathbf{h} = (\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H\mathbf{y} \tag{11.21}$$

where we have assumed $\mathbf{X}$ to be a full rank matrix.

COMMENT: you may be wondering whether some input sequences are better than others, and the answer is yes. If you consider the sequence $x(n)$ as the realization of a WSS process, chosen in a very vast class of WSS processes, it can be shown that optimal performances are achieved when the process is white. This is why in the practical problems of filter identification, the learning sequences used are as often as possible chosen "similar" to white noise.

Equation 11.21 should be compared with the result from exercise 8.5. If we multiply and divide by N, we can also write:

$$\mathbf{h} = \left(\frac{1}{N}\mathbf{X}^H\mathbf{X}\right)^{-1}\left(\frac{1}{N}\mathbf{X}^H\mathbf{y}\right) = \hat{\mathbf{R}}^{-1}\hat{\mathbf{r}} \tag{11.22}$$

an expression where $\hat{\mathbf{R}}$ is written as an input autocovariance matrix and $\hat{\mathbf{r}}$ as an output/input covariance vector of the filter. We will encounter later on an expression similar to 11.22 when we discuss the Wiener filtering.

Implementing the recursive least squares algorithm

The following program implements the recursive least squares algorithm to estimate an impulse response:

```
%===== IDENTIFRLS.M
SNR=10; h0=[1;-5/2;1]; P=length(h0);
N=200; x=randn(N,1); sigmab=10^(-SNR/20);
%=====
yssb=filter(h0,1,x);       % Output
y=yssb+sigmab*randn(N,1);  % Noised output
%===== RLS Algorithm
Qn=10^7*eye(P);            % Initialization
hn=zeros(P,1); xnp1=zeros(P,1);
for k=P:N
    xnp1(:)=x(k:-1:k-P+1);
    Qnh=Qn*xnp1; cc=1+xnp1'*Qnh; Kn=Qnh/cc;
    en(k)=(y(k)-xnp1'*hn); hn=hn+Kn*en(k);
    dh(k-P+1)=(hn-h0)'*(hn-h0);
    Qn=Qn-(Qnh*xnp1'*Qn/cc);
end
plot(10*log10(dh)); grid
%===== Theoretical limit
limT=10*log10(P/N)-SNR;
hold on; plot([0 N-P+1],[limT limT]); hold off
```

Figure 11.6 shows the evolution in decibels of the difference between the "true" filter and the identified filter, at each iteration step for different values of the signal-to-noise ratio.

You can check that the obtained result is almost identical to the one found with the direct calculation $\mathbf{h} = \mathbf{X}^{\#}\mathbf{y}$, using the pseudo-inverse and obtained with the following program:

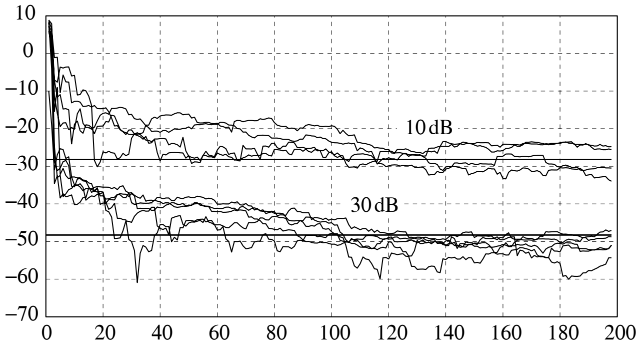


Figure 11.6 – *Difference between the exact response and the obtained response in dB*

```
X=toeplitz(x,[x(1) zeros(1,2)]);
h=X\y
```

The slight difference is due to the choice for the initial conditions of the recursive algorithm.

11.3 Linear predictions of the WSS processes

11.3.1 Yule-Walker equations

The *Yule-Walker* equations, laid down on page 308, relate the parameters (a_1, ..., a_p, σ^2) of an AR process defined by the recursive equation 8.53 to its covariance coefficients. As we are going to see, these equations are also the ones that relate the covariance coefficients of any WSS random process to the linear prediction coefficients.

Consider a *zero-mean*, WSS random process $x(n)$. The general expression of the process's *linear predictor* $\widehat{x}(n)$ at the time n, constructed from its N past values, is of the type:

$$\widehat{x}(n) = \alpha_1 x(n-1) + \cdots + \alpha_N x(n-N) = \sum_{i=1}^{N} \alpha_i x(n-i) \tag{11.23}$$

Theorem 11.1 states that the coefficients α_1, ..., α_N that minimize the *mean square error* $\mathbb{E}\left\{|x(n) - \widehat{x}(n)|^2\right\}$, *between the actual value* $x(n)$ and the *predicted value* $\widehat{x}(n)$ are such that the error:

$$e(n) = x(n) - \widehat{x}(n) \tag{11.24}$$

is orthogonal to any $x(n-k)$ with $1 \leq k \leq N$. This can be written:

$$\mathbb{E}\{(x(n) - \widehat{x}(n))x^*(n-k)\} = 0 \text{ for } 1 \leq k \leq N \tag{11.25}$$

Using the expectation's linearity, we get:

$$\mathbb{E}\{x(n)x^*(n-k)\} - \mathbb{E}\{\widehat{x}(n)x^*(n-k)\} = 0$$

By replacing $\widehat{x}(n)$ by its expression 11.23, then by again using the expectation's linearity, we get:

$$\mathbb{E}\{x(n)x^*(n-k)\} - \sum_{i=1}^{N} \alpha_i \mathbb{E}\{x(n-i)x^*(n-k)\} = 0$$

Because the process is stationary, $\mathbb{E}\{x(n)x^*(n-k)\} = R(k)$, and:

$$R(k) - \sum_{i=1}^{N} \alpha_i R(k-i) = 0 \text{ for } 1 \leq k \leq N \tag{11.26}$$

The minimum mean square error is given by 11.2 which here has the expression:

$$\varepsilon^2 = \mathbb{E}\{|e(n)|^2\} = \mathbb{E}\{(x(n) - \widehat{x}(n))x^*(n)\} = R(0) - \sum_{i=1}^{P} \alpha_i R(-i) \tag{11.27}$$

Stacking 11.27 and the N equations 11.26 in matrix form leads us to:

$$\begin{bmatrix} R(0) & R(-1) & \cdots & R(-N) \\ R(1) & R(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & R(-1) \\ R(N) & \cdots & R(1) & R(0) \end{bmatrix} \begin{bmatrix} 1 \\ -\alpha_1 \\ \vdots \\ -\alpha_N \end{bmatrix} = \begin{bmatrix} \varepsilon^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{11.28}$$

These equations are called the *Yule-Walker equations*, or the *normal equations*, and are the same as the equations 8.58 laid down in page 308. They allow us to calculate the prediction coefficients and of the minimum prediction mean-square error using the covariance matrix of a WSS process. In practice, the exact covariances can be replaced by their estimates according to expression 8.32. Remember that we gave in section 8.5.3 a fast algorithm for solving 11.28, the Levinson algorithm.

We are now going to see how these prediction coefficients can be expressed in the particular cases of a harmonic process and of an AR process.

11.3.2 Predicting a WSS harmonic process

We are first going to prove the following property, analogous to the one given on page 358 of Chapter 10 for deterministic signals:

Property 11.2 *Consider the WSS harmonic process defined by:*

$$x(n)=\sum_{k=1}^{P}\alpha_k e^{2j\pi f_k n} \tag{11.29}$$

where $\{\alpha_k\}$ is a sequence of P zero-mean uncorrelated complex variables, with the respective variances σ_k^2 and $\{f_k\}$ a sequence of P frequencies.

1. *There exist $b_1, \dots, b_P$ such that:*

$$x(n)+b_1x(n-1)+\cdots+b_Px(n-P)=0 \tag{11.30}$$

2. *The polynomial $B(z)=z^P+b_1z^{P-1}+\cdots+b_P$ has its P roots on the unit circle.*

Conversely, if $x(n)$ verifies 11.30 and if $B(z)$ has all its simple roots located on the unit circle, then $x(n)$ is of the type 11.29 where the α_k are P zero-mean, uncorrelated, complex random variables with any variances.

– To prove this, let $z_k=e^{2j\pi f_k}$ and $B(z)=\prod_{k=1}^{P}(z-z_k)$. With these notations we have on one hand $x(n)=\sum_{k=1}^{P}\alpha_k z_k^n$ and on the other hand the fact that the P degree polynomial $B(z)$ is such that $B(z_k)=0$. The polynomial $B(z)$ can also be developed as $B(z)=z^P+b_1z^{P-1}+\cdots+b_P$. We will now show that the obtained coefficients b_k are such that $x(n)+b_1x(n-1)+\cdots+b_Px(n-P)=0$. This is because we have:

$$\begin{aligned}&x(n)+b_1x(n-1)+\cdots+b_Px(n-P)\\&\quad=\sum_{k=1}^{P}\alpha_k z_k^n+b_1\sum_{k=1}^{P}\alpha_k z_k^{n-1}+\cdots+b_P\sum_{k=1}^{P}\alpha_k z_k^{n-P}\\&\quad=\sum_{k=1}^{P}\alpha_k z_k^{n-P}\left(z_k^P+b_1z_k^{P-1}+\cdots+b_P\right)=\sum_{k=1}^{P}\alpha_k z_k^{n-P}B(z_k)\end{aligned}$$

Therefore, the random variable $x(n)+b_1x(n-1)+\cdots+b_Px(n-P)$ is a linear combination of P random variables α_k, each of them multiplied by $z_k^{n-P}B(z_k)$ which is equal to 0. This proves it is null.

– Conversely, let us assume that $x(n)$ verifies 11.30. In that case, because the set of solutions is a P dimension subspace, we only need to find P solutions. We are going to find the ones of the type $x(n)=\alpha e^{2j\pi fn}$, where α and f are to be determined. By replacing this solution type in 11.30, and by assuming that $b_0=1$, we get, after simplifying by α, the equation:

$$\sum_{k=0}^{P}b_k e^{2j\pi f(n-k)}=e^{2j\pi f(n-P)}\sum_{k=0}^{P}b_k e^{2j\pi fk}e^{2j\pi f(n-P)}B(e^{2j\pi f})=0$$

> But by hypothesis, $B(z)$ has P distinct roots on the unit circle. This gives us the P solutions we were looking for. We still have to check that any linear combination of these solutions, of the type $\sum_k \alpha_k \exp(2j\pi f_k n)$, is a WSS process. All we need to do is choose P zero-mean uncorrelated complex variables, with any variances, as the α_k.

After an obvious notation change, the recursive equation $x(n) + b_1 x(n-1) + \cdots + b_P x(n-P) = 0$ can be rewritten as $x(n) = \beta_1 x(n-1) + \cdots + \beta_P x(n-P)$. This result is expressed as follows.

Property 11.3 (Linear prediction of a harmonic process) *Let $x(n)$ be the following harmonic process:*

$$x(n) = \sum_{k=1}^{P} \alpha_k e^{2j\pi f_k n}$$

where $\{\alpha_k\}$ is a sequence of P zero-mean, uncorrelated, complex random variables with the respective variances σ_k^2, and $\{f_k\}$ a sequence of P frequencies. Then for any $N \geq P$, the N-th order prediction coefficients $\beta_1, \ldots, \beta_N$ are given by:

$$\beta_i = \begin{cases} -b_i & \text{for} \quad 1 \leq i \leq P \\ 0 & \text{for} \quad P < i \leq N \end{cases}$$

where the b_i are the coefficients of the polynomial $B(z) = \prod_{k=1}^{P}(z - e^{2j\pi f_k})$ and the prediction mean square error is equal to 0.

As you can see, a process can be exactly predicted from its past. This is what is meant by the term "almost deterministic", used for such random processes.

The recursive equation $x(n) + b_1 x(n-1) + \cdots + b_P x(n-P) = 0$, given in property 10.1, should be compared with equation 8.53 defining a P-order AR random process but the second member of which would be null and the poles of which would be on the unit circle.

The previous results, obtained for a complex harmonic process, are still true for a process $x(n)$ that is the sum of P real sines of the type:

$$x(n) = \sum_{k=1}^{P} A_k \cos(2\pi f_k n + \Phi_k)$$

where $\{\Phi_k\}$ refers to a sequence of independent uniform random variables on $(0, 2\pi)$ independent of the A_k. We can rewrite:

$$x(n) = \sum_{k=1}^{P} \frac{A_k}{2}\left(e^{j\Phi_k} e^{2j\pi f_k n} + e^{-j\Phi_k} e^{-2j\pi f_k n}\right) = \sum_{k=1}^{2P} \alpha_k \zeta_k^n$$

where the ζ_k are $2P$ values of the type $e^{\pm 2j\pi f_k}$ and where the α_k are of the type $A_k \exp(\pm j\Phi_k)$ and are zero-mean uncorrelated r.v. with variances σ_k^2. Hence $x(n)$ appears as the sum of $2P$ complex exponentials the frequencies of which come in pairs of a positive and a negative value. Because of property, 11.2, $x(n)$ obeys a recursive equation of the type $x(n)+b_1x(n-1)+\cdots+b_{2P}x(n-2P)=0$ where the $2P$ degree polynomial has all of its roots located on the unit circle, come in pairs of complex conjugates. Therefore $B(z)$ has real coefficients.

11.3.3 Predicting a causal AR-P process

Property 11.4

Let $x(n)$ be a P order real AR process defined by equation 8.53 the expression of which is recalled here:

$$x(n) + a_1x(n-1) + \cdots + a_Px(n-P) = w(n)$$

with $A(z) \neq 0$ for $|z| \geq 1$ and where $w(n)$ is a WSS white process with the variance σ^2. Then for any $N \geq P$, the prediction coefficients $\alpha_1, \ldots, \alpha_N$ to the N-th order are given by:

$$\alpha_i = \begin{cases} -a_i & \textit{for} \quad 1 \leq i \leq P \\ 0 & \textit{for} \quad P < i \leq N \end{cases}$$

and the minimum prediction mean square error is equal to σ^2.

You can see this by rewriting equation 8.53 as follows:

$$x(n) = -a_1x(n-1) - \cdots - a_Px(n-P) + w(n) = \widetilde{x}(n) + w(n) \quad (11.31)$$

where we have defined:

$$\widetilde{x}(n) = -a_1x(n-1) - \cdots - a_Px(n-P) \quad (11.32)$$

For any $N \geq P$, $\widetilde{x}(n)$ coincides with the best linear estimation given by 11.23. Indeed, for any $k \geq 1$:

$$\mathbb{E}\{w(n)x^*(n-k)\} = \mathbb{E}\{w(n)\}\mathbb{E}\{x^*(n-k)\} = 0 \quad (11.33)$$

This result simply expresses the fact that $x(n-k)$ is a function only of $w(n-k)$, $w(n-k-1)\ldots$ since the solution is causal and $w(n)$ is white. Therefore $x(n-k)$ and $w(n)$ are not correlated. We infer 11.33 from this result. By replacing, according to 11.31, $w(n)$ by $x(n) - \widetilde{x}(n)$ in 11.33, we get:

$$\mathbb{E}\{(x^*(n) - \widetilde{x}^*(n))x(n-k)\} = 0 \text{ for } \forall k \geq 1$$

This relation is identical to 11.25. It proves that $\widetilde{x}(n)$ is the best linear prediction of $x(n)$, in terms of the minimum mean square error, calculated on any past with a duration higher than P. Notice that it only uses its last P values. In other words, for a P-order AR, the orthogonal projection on the entire past coincides with the orthogonal projection on the last P instants. Still another way of saying it is that for an AR model, the prediction coefficients coincide with the model's parameters. This result is usually false for any WSS process, for example an MA. This should not stop us, however, from looking into the prediction of these processes.

Furthermore, the white input process $w(n)$ appears as the *prediction error* and its variance as the minimum mean square error.

We also should mention that in the field of statistics, the name *linear regression* is also used for referring to linear prediction. Hence the name *autoregressive* given to these processes.

11.3.4 Reflection coefficients and lattice filters

When discussing the Burg algorithm, we presented two types of errors called *forward* and *backward*. We are going to describe them more precisely as well as the reflection coefficient k_m defined by 8.69. When linearly predicting x_n as a function of $x_{n-1}, \ldots, x_{n-m+1}$ we write that the error prediction (equation 11.24) is orthogonal to the past[2], that is:

$$\varepsilon^F_{m-1}(n) = \left(x(n) - \sum_{k=1}^{m-1} \alpha_k(m-1)x(n-k)\right) \perp \{x(n-1), \ldots, x(n-m+1)\}$$

The exponent F (as in forward) indicates a prediction done from the past to the present, and the index $m-1$ indicates that the prediction is done based on $(m-1)$ past values. Notice that $\varepsilon^F_{m-1}(n)$ is a linear combination of x_n, $x_{n-1}, \ldots, x_{n-m+1}$, and that because of stationarity, the values of $\alpha_k(m-1)$ that minimize the square deviation does not depend on n.

We are now going to perform what is called the *backward* "prediction" of x_{n-m} as a function of the $(m-1)$ values of the immediate future, that is $x_{n-m+1}, \ldots, x_{n-1}$. We have:

$$\begin{aligned} \varepsilon^B_{m-1}(n-1) &= x(n-m) - \sum_{k=1}^{m-1} \beta_k(m-1)x(n-m+k) \\ &\quad \perp \{x(n-m+1), \ldots, x(n-1)\} \end{aligned}$$

$\varepsilon^B_{m-1}(n-1)$ is a linear combination of $x_{n-1}, x_{n-2}, \ldots, x_{n-m}$, hence the index $(n-1)$ in its definition. Notice that, because of stationarity, $\varepsilon^B_{m-1}(n) = x(n+1-m) - \sum_{k=1}^{m-1} \beta_k(m-1)x(n+1-m+k)$ with a sequence of coefficients $\beta_k(m-1)$ that do not depend on n.

[2] In this paragraph, we assume the processes to be real and omit the star indicating conjugation.

By expressing the orthogonality, and by using the stationarity hypothesis, we can easily check that the same Yule-Walker equations are obtained for the *forward* and *backward* predictions and that:

$$\alpha_k(m-1) = \beta_k(m-1) \tag{11.34}$$

Notice that for $m = 0$, the *Forward* and *Backward* prediction errors are written:

$$\varepsilon_0^F(n) = x(n) \quad \text{and} \quad \varepsilon_0^B(n) = x(n) \tag{11.35}$$

respectively. By construction, $\varepsilon_{m-1}^F(n)$ and $\varepsilon_{m-1}^B(n-1)$ are orthogonal to the *same* subspace generated by x_{n-1}, x_{n-2}, $\ldots$, x_{n-m+1}, and therefore for any scalar λ:

$$\delta \quad = \quad \varepsilon_{m-1}^F(n) + \lambda\varepsilon_{m-1}^B(n-1) \perp \{x_{n-1}, x_{n-2}, \ldots, x_{n-m+1}\}$$

Let us choose λ such that δ is also orthogonal to x_{n-m}, and refer to this particular value of λ as k_m. The vector $\delta_m = \varepsilon_{m-1}^F(n) + k_m\varepsilon_{m-1}^B(n-1)$ is therefore orthogonal to the subspace generated by x_{n-1}, x_{n-2}, $\ldots$, x_{n-m}. Because δ_m is a linear combination of the type $x(n) - u(n)$ where $u(n)$ is a linear combination of x_{n-1}, x_{n-2}, $\ldots$, x_{n-m}, δ_m is, according to the projection theorem, the m-th step prediction error. Hence we can write it:

$$\varepsilon_m^F(n) = \varepsilon_{m-1}^F(n) + k_m\varepsilon_{m-1}^B(n-1)$$

where k_m is defined by:

$$\varepsilon_{m-1}^F(n) + k_m\varepsilon_{m-1}^B(n-1) \perp x(n-m)$$

which is expressed as $\mathbb{E}\left\{\varepsilon_{m-1}^F(n)x(n-m)\right\} + k_m\mathbb{E}\left\{\varepsilon_{m-1}^B(n-1)x(n-m)\right\} = 0$.

By replacing the prediction errors with their expressions as functions of $x(n)$, we get:

$$k_m = -\frac{R(m) - \sum_{k=1}^{m-1}\alpha_k(m-1)R(m-k)}{R(0) - \sum_{k=1}^{m-1}\alpha_k(m-1)R(k)}$$

where $R(k)$ refers to the covariance function of the WSS process $x(n)$. This proves expressions 9.14 and 8.69. Expression 9.15, which is recalled below, can be demonstrated in the same way:

$$\epsilon_m^B(n) = \epsilon_{m-1}^B(n-1) + k_m\epsilon_{m-1}^{F*}(n)$$

The analysis filter: $x(n) \mapsto \varepsilon_m^F(n)$

We are now going to construct a filter with the input $x(n)$ and the two prediction errors as the outputs. We need to start with equations 9.14 and 9.15:

$$\left\{\begin{array}{l} \varepsilon_m^F(n) = \varepsilon_{m-1}^F(n) + k_m \varepsilon_{m-1}^B(n-1) \\ \varepsilon_m^B(n) = \varepsilon_{m-1}^B(n-1) + k_m \varepsilon_{m-1}^F(n) \end{array}\right. \tag{11.36}$$

Relations 11.36 lead to the simplified representation 11.7of the block where the two input sequences are $\varepsilon_{m-1}^F(n)$ and $\varepsilon_{m-1}^B(n)$ and the two output sequences are $\varepsilon_m^F(n)$ and $\varepsilon_m^B(n)$.

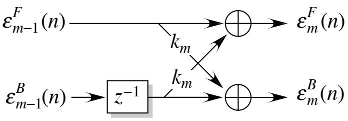


Figure 11.7 – *Modulus of the lattice analysis filter*

If we cascade these blocks and use the initial condition $\varepsilon_0^F(n) = \varepsilon_0^B(n) = x(n)$, we get the filtering diagram of Figure 11.8, called the *lattice filter*, which changes the signal $x(n)$ into the prediction errors.

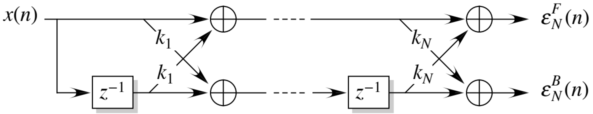


Figure 11.8 – *The lattice analysis filter*

We are now going to study the transfer function's expression. Starting with:

$$\left\{\begin{array}{l} \varepsilon_m^F(n) = x(n) - \sum_{k=1}^m \alpha_k(m) x(n-k) \\ \varepsilon_m^B(n) = x(n-m) - \sum_{k=1}^m \beta_k(m) x(n-m+k) \end{array}\right.$$

we can denote by $G_m^F(z)$ the FIR filter with $x(n)$ as the input and $\varepsilon_m^F(n)$ as the output. This filter has the transfer function $G_m^F(z) = 1 - \sum_{k=1}^m \alpha_k(m) z^{-k}$. Likewise, $G_m^B(z) = z^{-m}(1 - \sum_{k=1}^m \beta_k(m) z^k)$ refers to the filter associated with $\varepsilon_m^B(n)$. Because $\alpha_k(m) = \beta_k(m)$, we have $G_m^B(z) = z^{-m} G_m^F(1/z)$. Using obvious notations, we can write $E_m^F(z) = G_m^F(z) X(z)$ and $E_m^B(z) = G_m^B(z) X(z)$.

We are going to prove by mathematical induction that all of the zeros of the filter $G_m^F(z)$ have a modulus smaller than 1. If we consider the z-transform of 11.36, we get:

$$\left\{\begin{array}{l} G_m^F(z) = G_{m-1}^F(z) + k_m z^{-1} G_{m-1}^B(z) \\ G_m^B(z) = z^{-1} G_{m-1}^B(z) + k_m G_{m-1}^F(z) \end{array}\right. \tag{11.37}$$

Let z_i be a zero of $G_m^F(z)$. According to the first equation of 11.37, we have $G_{m-1}^F(z_i) + k_m z_i^{-1} G_{m-1}^B(z_i) = 0$. This can be written:

$$|P(z_i)| = \frac{|z_i|}{|k_m|}$$

where we have assumed:

$$P(z) = \frac{G_{m-1}^B(z)}{G_{m-1}^F(z)} = \frac{a_{m-1} z^{(m-1)} + \cdots + a_1 z + 1}{z^{m-1} + \cdots + a_{m-2} z + a_{m-1}} = \prod_{k=1}^{m-1} \frac{1 - b_k z}{z - b_k}$$

with $a_k = -\alpha_k(m-1)$. We then check that if b_k is one of the denominator's roots, then $1/b_k$ is one of the numerator's roots. If we assume that all of the roots of $G_{m-1}^F(z)$ have a modulus strictly smaller than 1, the rational function $P(z)$ is an all-pass filter and as such verifies theorem 4.5. In particular, $|P(z)| < 1$ for $|z| > 1$. But because $|P(z_i)| = |z_i|/|k_m|$, and because of inequality 8.71, $|k_m| < 1$, $|P(z_i)| > |z_i|$. This means that $|z_i| < 1$. In the case where $k_m = 1$, we know that the process is harmonic (the prediction error is null beyond a certain point), and the roots of $G_m^F(z)$ are all on the unit circle. Its inverse has a causal and stable representation.

The synthesis filter: $\varepsilon_m^F(n) \mapsto x(n)$

The filter that changes $\varepsilon_m^F(n)$ into $x(n)$ is causal and stable. Starting off with equations 11.36, we can write:

$$\left\{\begin{array}{l} \varepsilon_{m-1}^F(n) = \varepsilon_m^F(n) - k_m \varepsilon_{m-1}^B(n-1) \\ \varepsilon_m^B(n) = \varepsilon_{m-1}^B(n-1) + k_m \varepsilon_{m-1}^F(n) \end{array}\right. \tag{11.38}$$

Relations 11.38 lead to a cell set-up with the input sequences $\varepsilon_m^F(n)$ and $\varepsilon_{m-1}^B(n)$ and the two output sequences $\varepsilon_{m-1}^F(n)$ and $\varepsilon_m^B(n)$. If we cascade these cells, we get the filtering diagram of Figure 11.9, called a *synthesis filter*,which changes the signal $\varepsilon_m^F(n)$ into the signal $x(n)$. The previous results guarantee that if $|k_m| < 1$, the obtained filter is stable.

The two functions `atok` and `ktoa` allow us to go from the coefficients a_i to the k_i and conversely:

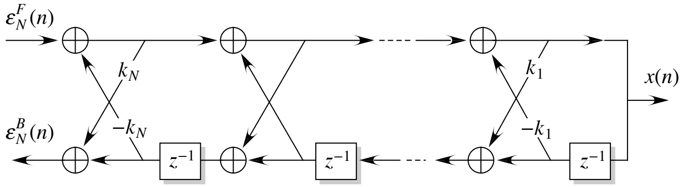


Figure 11.9 – *The lattice synthesis filter*

```
function ki=atok(ai)
%%===============================================%
%% From the AR parameters (ai) to the reflection %
%% coefficients (ki)                             %
%% SYNOPSIS: ki=ATOK(ai)                         %
%%    ai = AR model (1 a1 ... aP)                %
%%    ki = reflection coefficients (k1 ... kP)   %
%%===============================================%
P=length(ai)-1; ki=zeros(P,1);
for ii=P:-1:1
    ki(ii)=ai(ii+1); bi=conj(ai(ii+1:-1:1));
    umodk2=1-ki(ii)*ki(ii)';
    ai=(ai-ki(ii)*bi)/umodk2;
    ai(ii+1)=[];
end
return
```

and conversely from the k_i to the a_i:

```
function ai=ktoa(ki)
%%=============================================%
%% From the reflection coefficients (ki)       %
%% to the AR parameters (ai)                   %
%% SYNOPSIS: ai=KTOA(ki)                       %
%%    ki = reflection coefficients (k1 ... kP) %
%%    ai = AR-model parameters (1 a1 ... aP)   %
%%=============================================%
P=length(ki); ai=[1;zeros(P,1)];
for ii=1:P
    ai(1:ii+1) = ai(1:ii+1) + ki(ii)*conj(ai(ii+1:-1:1));
end
return
```

Exercise 11.3 (Lattice filtering)

1. Analysis: write the lattice filtering function that implements the formulae 11.36.

2. Synthesis: write the lattice filtering function that implements the formulae 11.38.

3. Write a program that checks the two previous functions by generating an AR-P process using white noise and the `filter` function. Use the `atok` function to extract the reflection coefficients.

11.4 Wiener filtering

Consider the diagram in Figure 11.10.

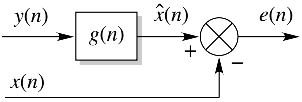


Figure 11.10 – *The Wiener filter*

Let $y(n)$ be the observation signal and $x(n)$ the desired signal. $x(n)$ and $y(n)$ are assumed to be zero-mean, WSS, real random processes with stationary covariances. Stationarity implies:

$$\begin{cases} \mathbb{E}\{x(n+k)x(n)\} = R_{xx}(k) \\ \mathbb{E}\{y(n+k)y(n)\} = R_{yy}(k) \\ \mathbb{E}\{x(n+k)y(n)\} = R_{xy}(k) \end{cases}$$

All of these sequences are assumed to be known.

We are going to determine the linear filter with the impulse response $g(n)$ that minimizes the positive quantity:

$$J(\{g(n)\}) = \mathbb{E}\left\{\varepsilon(n)^2\right\} = \mathbb{E}\left\{|x(n) - \hat{x}(n)|^2\right\} \tag{11.39}$$

where:

$$\hat{x}(n) = \sum_k g(k)y(n-k)$$

The projection theorem gives us the solution. If we use the same notations as in paragraph 11.1, $\mathcal{H}$ is the space of square summable random variables and $\mathcal{C}$ is the subspace generated by linear combinations of elements of the sequence $\{y(n)\}$ written $\hat{x}(n) = \sum_k g(k)y(n-k)$.

The filter that minimizes the square deviation between $x(n)$ and $\hat{x}(n)$ is such that the difference $x(n) - \hat{x}(n)$ is orthogonal to any element belonging to the space $\mathcal{C}$. Hence, in terms of orthogonality in the Hilbert space of square summable random variables, we have for any k:

$$\mathbb{E}\{[x(n) - \hat{x}(n)]y(n-k)\} = \mathbb{E}\{[x(n) - \textstyle\sum_m g(k)y(n-m)]y(n-k)\} = 0$$

If we develop and use the stationarity hypotheses of $y(n)$ we get:

$$R_{xy}(p) = \sum_k g(k)R_{yy}(p-k) \tag{11.40}$$

This equation is called the *Wiener equation* and the solution is called the *Wiener filter.* It is in agreement with the expression 11.22 we saw in the case of channel identification.

Unconstrained solution in $\mathcal{C}$

Let us first consider the case where $\mathcal{C}$ is the set of linear combinations of all the variables of the sequence $\{y(n)\}$: the summation in equation 11.40 is then performed from $-\infty$ to $+\infty$. Hence we have a convolution of $g(k)$ with $R_{yy}(k)$.

To solve equation 11.40, we can then use the DTFT. If we denote by $S_{yy}(f)$ and $S_{xy}(f)$ the respective DTFTs of $R_{yy}(k)$ and $R_{xy}(k)$, and if we use the convolution properties, we get:

$$G(f) = \frac{S_{xy}(f)}{S_{yy}(f)} \tag{11.41}$$

Expression 11.41 is usually in the form of a rational function with poles inside and outside the unit circle. The stable solution is then bilateral. We can, however, extract by truncation a causal approximation if we tolerate a certain delay.

Imposing time constraints

The following three cases show some practical interest, but unfortunately they cannot be solved as easily as expression 11.41:

- $\mathcal{C}$ is the set of linear combinations of $y(k)$ with $k \leq n$ up to the present time n. In this case:

  $$\hat{x}(n) = \sum_{k=-\infty}^{n} g(n-k)y(k) = \sum_{m=0}^{+\infty} g(m)y(n-m)$$

  This is called *filtering.*

- $\mathcal{C}$ is the set of linear combinations of $y(k)$ with $k \leq (n-p)$, $p > 0$, up to the past time $(n-p)$. In this case:

  $$\hat{x}(n) = \sum_{k=-\infty}^{n-p} g(n-k)y(k) = \sum_{k=p}^{+\infty} g(k)y(n-k)$$

  This is called *prediction.*

- $\mathcal{C}$ is the set of linear combinations of $y(k)$ with $k \leq (n+p)$, $p > 0$, up to the future time $(n+p)$. In this case:

$$\hat{x}(n) = \sum_{k=-\infty}^{n+p} g(n-k)y(k) = \sum_{k=-p}^{+\infty} g(k)y(n-k)$$

  This is called *smoothing.*

In the case of filtering, $\mathcal{C}$ is the set of linear combinations of all the variables $y(n)$ for $n \in \{-\infty, \cdots, 0\}$ and equation 11.40 can be written:

$$R_{xy}(p) = \sum_{k=0}^{\infty} g(k)R_{yy}(p-k)$$

Because the summation starts at 0, it cannot be solved simply by calculating the DTFT of the two sides. Performing the calculation without taking any precautions does lead to a causal solution but one that is not stable. The right solution was found by Wiener. Its expression goes beyond the aim of this book, but can be found in [84].

11.4.1 Finite impulse response solution

Imagine that we are trying to find as a solution for $g(n)$ an *FIR* filter with a length $N = A + C$, denoted by:

$$g(-A), \ldots, g(-1), g(0), \ldots, g(C-1)$$

A is used here to take into account a certain delay. Using equation 11.40 leads us to the expression:

$$\begin{aligned} R_{xy}(p) &= g(-A)R_{yy}(p+A) + \cdots + g(-1)R_{yy}(p+1) + \\ &\qquad g(0)R_{yy}(p) + g(1)R_{yy}(p-1) + \cdots + g(C-1)R_{yy}(p-C+1) \end{aligned}$$

If we stack the $N = A + C$ expressions of $R_{xy}(p)$ for p from 0 to $N-1$, we get the matrix expression:

$$\mathbf{Rg} = \mathbf{r} \tag{11.42}$$

where:

$$\mathbf{R} = \begin{bmatrix} R_{yy}(0) & \cdots & R_{yy}(-N+1) \\ \vdots & \ddots & \vdots \\ R_{yy}(N-1) & \cdots & R_{yy}(0) \end{bmatrix} \quad \text{and} \quad \mathbf{r} = \begin{bmatrix} R_{xy}(-A) \\ \vdots \\ R_{xy}(-1) \\ R_{xy}(0) \\ \vdots \\ R_{xy}(C-1) \end{bmatrix}$$

If $\mathbf{R}$ is invertible, the solution then has the expression:

$$\mathrm{g} = \mathbf{R}^{-1}\mathbf{r} \tag{11.43}$$

It should be compared to expression 11.22.

Generally speaking, you can see that finding the solution requires the inversion of the matrix $\mathbf{R}$. It can however be avoided by implementing the *gradient algorithm*.

11.4.2 Gradient algorithm

A general and simple idea for calculating a minimum with respect to g of the function $J(\mathrm{g})$ is to calculate $\mathrm{g}(n)$ at the n-th step using the value of $\mathrm{g}(n-1)$ at the $(n-1)$-th step, going in the direction opposite to that of the gradient. This can be written as follows:

$$\mathrm{g}(n) = \mathrm{g}(n-1) - \frac{\mu}{2}\left.\frac{\partial J(\mathrm{g})}{\partial \mathrm{g}}\right|_{\mathrm{g}=\mathrm{g}_{n-1}} \tag{11.44}$$

where μ is a *positive* scalar called the *gradient step.*

If the function $J(\mathrm{g})$ is regular enough and if μ is small enough, $\mathrm{g}(n)$ should converge and hence the resulting convergence value obeys equation 11.44. Therefore, it *sets the gradient to zero*, which makes it a good candidate for the search for a minimum.

Let us apply this result to equation 11.39. If we use the fact that $\mathrm{g}(n)$ has a finite length, we get:

$$\begin{aligned} J(\mathrm{g}) &= \mathbb{E}\{(x(n) - \mathrm{g}^T\mathbf{y}(n))(x(n) - \mathbf{y}^T(n)\mathrm{g})\} \\ &= R_{xx}(0) - 2\mathrm{g}^T\mathbf{r} + \mathrm{g}^T\mathbf{R}\mathrm{g} \end{aligned}$$

the gradient of which with respect to g has the expression:

$$\frac{\partial J(\mathrm{g})}{\partial \mathrm{g}} = -2(\mathbf{r} - \mathbf{R}\mathrm{g}) \tag{11.45}$$

By replacing this expression, calculated at the point $\mathrm{g} = \mathrm{g}(n-1)$, in equation 11.44, we get the following recursive equation:

$$\mathrm{g}(n) = \mathrm{g}(n-1) - \mu\,(\mathbf{R}\mathrm{g}(n-1) - \mathbf{r}) \tag{11.46}$$

called the *gradient algorithm.*

Study of equation 11.46

We are going to determine the expression of the solution $\mathbf{g}(n)$ to equation 11.46 as a function of the initial value $\mathbf{g}(0)$ and of the matrix $\mathbf{A} = \mathbf{I} - \mu\mathbf{R}$. In order to do this, we need to write the recurrent equations for the first n values:

$$\begin{array}{rcl|cl} \mathbf{g}(n) & = & (\mathbf{I} - \mu\mathbf{R})\mathbf{g}(n-1) + \mu\mathbf{r} & \times & \mathbf{I} \\ \mathbf{g}(n-1) & = & (\mathbf{I} - \mu\mathbf{R})\mathbf{g}(n-2) + \mu\mathbf{r} & \times & (\mathbf{I} - \mu\mathbf{R}) \\ & \vdots & & \vdots & \\ \mathbf{g}(1) & = & (\mathbf{I} - \mu\mathbf{R})\mathbf{g}(0) + \mu\mathbf{r} & \times & (\mathbf{I} - \mu\mathbf{R})^{n-1} \end{array}$$

If both sides are multiplied on the left by the indicated quantities, we get:

$$\begin{aligned} \mathbf{g}(n) &= \mu(\mathbf{I} + \cdots + \mathbf{A}^{n-1})\mathbf{r} + \mathbf{A}^n\mathbf{g}(0) \\ &= \mu(\mathbf{I} - \mathbf{A}^n)(\mathbf{I} - \mathbf{A})^{-1}\mathbf{r} + \mathbf{A}^n\mathbf{g}(0) \\ &= (\mathbf{I} - \mathbf{A}^n)\mathbf{R}^{-1}\mathbf{r} + \mathbf{A}^n\mathbf{g}(0) \end{aligned} \qquad (11.47)$$

where we have imposed $\mathbf{A} = (\mathbf{I} - \mu\mathbf{R})$ and used the identity $(\mathbf{I} + \cdots + \mathbf{A}^{n-1})(\mathbf{I} - \mathbf{A}) = \mathbf{I} - \mathbf{A}^n$.

A classic result states that if the eigenvalues of $\mathbf{A}$ have a modulus smaller than 1, the matrix $\mathbf{A}^n$ tends to the zero matrix when n tends to infinity. But since any eigenvector of $\mathbf{R}$ associated with the eigenvalue λ_i is an eigenvector of $\mathbf{A}$ associated with the eigenvalue $(1 - \lambda_i\mu)$, the stability condition can be written $-1 < 1 - \lambda_i\mu < 1$. Because λ_i is positive (eigenvalue of a covariance matrix), $0 < \mu < 2/\lambda_i$. Therefore, if μ is such that:

$$0 < \mu < \frac{2}{\max_i(\lambda_i)}$$

then the algorithm converges and the convergence solution, denoted by $\mathbf{g}_\infty$, obeys the recurrent equation. This means that $\mathbf{g}_\infty = \mathbf{g}_\infty + \mu(R\mathbf{g}_\infty - \mathbf{r})$, which leads to $\mathbf{R}\mathbf{g}_\infty = \mathbf{r}$, which is precisely expression 11.42, $\mathbf{R}\mathbf{g}_w = \mathbf{r}$, leading to the Wiener filter $\mathbf{g}_w$.

The *misadjustment* is defined by the expression:

$$\Delta_n = J(\mathbf{g}(n)) - J_{\min} \qquad (11.48)$$

where $J_{\min}$ refers to the criterion value with the optimal filter. First we must determine the criterion at the n-th step. We have:

$$\begin{aligned} J(\mathbf{g}(n)) &= R_{xx}(0) - 2\mathbf{g}^T(n)\mathbf{r} + \mathbf{g}^T(n)\mathbf{R}\mathbf{g}(n) \\ &= R_{xx}(0) - 2\mathbf{g}^T(n)\mathbf{R}\mathbf{g}_w + \mathbf{g}^T(n)\mathbf{R}\mathbf{g}(n) \end{aligned}$$

The criterion value with the optimal filter $\mathbf{g}_w$ is given by:

$$J_{\min} = R_{xx}(0) - 2\mathbf{r}^T\mathbf{g}_w + \mathbf{g}_w^T\mathbf{R}\mathbf{g}_w = R_{xx}(0) - \mathbf{g}_w^T\mathbf{R}\mathbf{g}_w$$

This means that the *misadjustment* has the expression:

$$\Delta_n = (\mathbf{g}(n) - \mathbf{g}_w)^T \mathbf{R} (\mathbf{g}(n) - \mathbf{g}_w)$$

Therefore, when $n \to +\infty$, the speed at which the misadjustment decreases to 0 is related to the distribution of the eigenvalues of $\mathbf{R}$.

Simulations

The following program uses simulation to evaluate the performances of the gradient algorithm. For $\mathbf{R}$ and $\mathbf{g}$ such that $\mathbf{r} = \mathbf{R}\mathbf{g}$, the square deviation is plotted against the number of iterations. The program shows that about a thousand iterations are necessary to obtain $\mathbf{g}_w$ with a good accuracy. The gradient algorithm is slow:

```
%===== SIMULGRADDETER.M
%===== Eigenvalues
lambda=[1 0.01 0.0001]; nvp=length(lambda);
R=diag(lambda); gw=ones(nvp,1); rxy=R*gw;
%=====  Initializations
lambda=[1 0.01 0.0001]; nlambda=length(lambda);
R=diag(lambda); gw=ones(nlambda,1); rxy=R*gw;
%=====
mu=1/2; N=100000;
d2g=zeros(N,1); gn=zeros(nlambda,1);
for n=1:N
    gn=gn+mu*(rxy-R*gn);
    d2g(n)=(gn-gw)'*(gn-gw);
end
semilogx(d2g); grid
```

Figure 11.11 shows the evolution of the square deviation over the course of $N = 100{,}000$ iterations. Notice that the graph shows three different parts corresponding to the three consecutive decompositions of the modes, corresponding themselves to the three eigenvalues of $\mathbf{R}$, from the smallest to the largest. This is a very general result. The determinating factor in the decreasing speed of the modes associated with each of the eigenvalues is the product $\mu\lambda_i$.

The speed increases as $\mu\lambda_i$ gets closer to 2 (from below). Hence, for the highest eigenvalue, the choice of $\mu = 1$ is the right one; however, for the smallest eigenvalue 0.0001, a much greater value of μ would be needed to accelerate the decrease. Unfortunately, we can not change the value of μ, otherwise the algorithm might diverge.

The conclusion is that the number of iterations leading to the solution is related to the smallest eigenvalue, and that the choice of the gradient step for convergence is associated with the highest eigenvalue.

A more subtle method would be to work separately on each of the three eigenvectors of the covariance matrix with three different steps better adapted

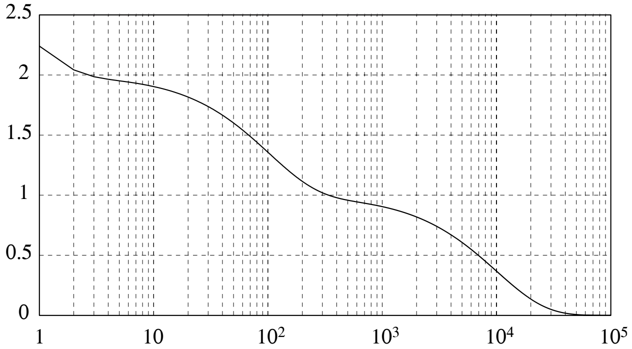


Figure 11.11 – *Evolution of the square deviation between the actual value of* $\mathbf{g}_w$ *and the obtained value, as a function of the number of iteration steps of the gradient algorithm*

to the three eigenvalues. This can be expressed as follows:

$$\mathbf{g}(n) = \mathbf{g}(n-1) - \begin{bmatrix} \mu_1 & 0 & 0 \\ 0 & \mu_2 & 0 \\ 0 & 0 & \mu_3 \end{bmatrix} (\mathbf{Rg}(n-1) - \mathbf{r})$$

In fact, as we are going to prove, the right matrix is the inverse of the covariance matrix. To understand this, consider the following algorithm:

$$\mathbf{g}_n = \mathbf{g}_{n-1} - \left[\frac{\partial^2 J(\mathbf{g})}{\partial \mathbf{g}^2}\right]^{-1} \left.\frac{\partial J(\mathbf{g})}{\partial \mathbf{g}}\right|_{\mathbf{g}=\mathbf{g}_{n-1}} \qquad (11.49)$$

in which $\mu\mathbf{I}$ was replaced with the inverse of the Hessian of $J(\mathbf{g})$. A simple calculation shows that $J(\mathbf{g})$ is equal to $2\mathbf{R}$. Therefore, at the n-th step and for any $\mathbf{g}(n-1)$:

$$\mathbf{g}(n) = \mathbf{g}(n-1) - \mathbf{R}^{-1}(-\mathbf{r} + \mathbf{Rg}(n-1)) = \mathbf{R}^{-1}\mathbf{r} = \mathbf{g}_w \qquad (11.50)$$

Hence the algorithm reaches its minimum in *one* step, meaning that the right matrix is not $\mu\mathbf{I}$, with μ having the same order of magnitude as the inverse of the eigenvalues of $\mathbf{R}$, but precisely $\mathbf{R}^{-1}$. The major drawback is that this algorithm loses its essential quality, that is to say its simplicity.

Comments

The criterion $J(\mathbf{g})$ is shaped like a bowl, the bottom of which corresponds to the Wiener solution we wish to reach. As μ increases, $J(\mathbf{g})$ takes bigger steps towards the bottom of the bowl. However, as it gets closer to the minimum, it is better to have a small value of μ if we want to get as close as possible. To give you an idea, a possible value for μ is $2/(\max_i(\lambda_i) + \min_i(\lambda_i))$.

The following program plots, against μ, for a given number n of iterations the square deviation in dB between the actual value $\mathbf{g} = \mathbf{R}^{-1}\mathbf{r}$ and the value given by expression 11.47 (for $\mathbf{g}(0) = 0$):

$$\mathbf{g}(n) = \mathbf{g}_w - \mathbf{A}^n\mathbf{R}^{-1}\mathbf{r} \tag{11.51}$$

You can see in Figure 11.12 that, for the values that were chosen, there is a minimum deviation in the neighborhood of the value $2/(\max_i(\lambda_i)+\min_i(\lambda_i)) = 1/(1+0.1)$ indicated previously:

```
%===== GRADMU1.M
%===== (lambda * mu) must be < 2
lambda=[1 0.1]; nvp=length(lambda);
R=diag(lambda); gw=ones(nvp,1);
rxy=R*gw;
%=====
N=50; mu=(1:0.02:1.99); L=length(mu);
desj=zeros(L,1);    % J(g) estimated
for ii=1:L
    mui=mu(ii);
    A=eye(nvp)-(eye(nvp)-mui*R)^N;
    gn=A*inv(R)*rxy;
    desj(ii)=10*log10((gn-gw)'*(gn-gw));
end
plot(mu,desj); grid
%===== A good value
mu0=2/(1+0.1);
A=eye(nvp)-(eye(nvp)-mu0*R)^N;
gn=A*inv(R)*rxy;
desj0=10*log10((gn-gw)'*(gn-gw));
hold on; plot(mu0,desj0,'o'); hold off
```

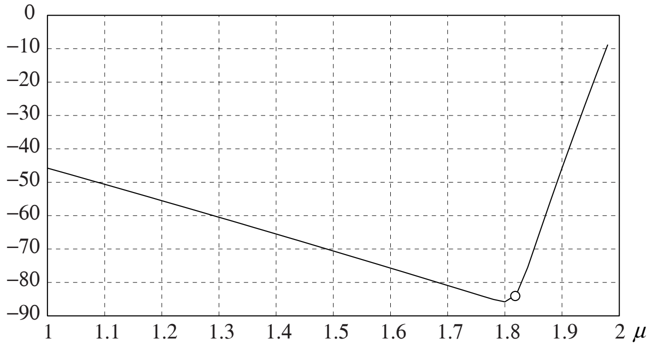


Figure 11.12 – *Square deviation in dB plotted against μ. The 'o' indicates the deviation obtained for the value $2/(\max_i(\lambda_i)+\min_i(\lambda_i))$*

Influence of the eigenvalues of R on the performances

We are now going to numerically study how the distribution of the eigenvalues of $\mathbf{R}$ influences the shape of the trajectories of $\mathbf{g}(n)$. We will assume that the Wiener solution is the filter with two coefficients $g_w(0) = 2$ and $g_w(1) = 6$. This is possible, for a given matrix, if we choose $\mathbf{r} = \mathbf{R}\mathbf{g}_w$.

We saw that convergence was related to the eigenvalues of $\mathbf{R}$. Remember that $\mathbf{R}$ is a covariance matrix (its size here is 2×2). Therefore, it has two positive eigenvalues. We are going to study changes in performance when the ratio ρ of its two eigenvalues varies. If we consider again the criterion's expression:

$$J(g_0, g_1) = R_{xx}(0) - 2[g_0 \quad g_1]\mathbf{r} + [g_0 \quad g_1]\mathbf{R}\begin{bmatrix} g_0 \\ g_1 \end{bmatrix}$$

where $\mathbf{R}$ is positive. This is a second degree polynomial in (g_0, g_1) that generates a paraboloid, the sections of which are ellipses. The eigendecomposition of $\mathbf{R}$ is as follows:

$$\mathbf{R} = \begin{bmatrix} R_y(0) & R_y(1) \\ R_y(1) & R_y(0) \end{bmatrix} = \mathbf{U}\begin{bmatrix} \sigma_0^2 & 0 \\ 0 & \sigma_1^2 \end{bmatrix}\mathbf{U}^T$$

where $\mathbf{U}$ is a 2×2 unitary matrix, the column vectors of which are the directions corresponding to the major and minor axes of the ellipse, and σ_0^2 and σ_1^2 are quantities that measure the ellipse's eccentricity. Let $\rho = \sigma_1^2/\sigma_0^2$ and let $\sigma_0^2 + \sigma_1^2 = 1$. This condition means that the power $y(n)$ is constrained to be constant, because $\mathbf{R}$ is the covariance matrix of $y(n)$, which is assumed to be stationary, and therefore the trace (sum of the diagonal elements) of $\mathbf{R}$ is equal to $2R_{yy}(0)$. But since the trace of a matrix is independent of the basis chosen for its decomposition, the trace is also equal to $\sigma_0^2 + \sigma_1^2 = 1$. This means we can make σ_0^2 and σ_1^2 parameters of our problem as functions of ρ, under the constraint of a constant power, using the two expressions:

$$\sigma_0^2 = \frac{1}{1+\rho} \text{ and } \sigma_1^2 = \frac{\rho}{1+\rho}$$

We start with a known initial value $\mathbf{g}(1)$, and as ρ varies, observe the shape of the trajectory described by $\mathbf{g}(n)$, which is calculated at each step n of the algorithm. This trajectory tends to the point with coordinates $\mathbf{g}_w$ corresponding to the Wiener solution. The initial value can vary.

We used the following program to carry out the numerical study:

```
%===== GRADDET22.M
clear; N=70;           % Number of iterations
gw=[2;6];              %  Wiener filter
ginit=[1;5];           % Initialization
rho=(2:5); lrho=length(rho);
gn=zeros(2,N); gnc=zeros(N,lrho);
```

```
mu=0.1;                 % Step
%===== Unitary matrix
theta=-3*pi/7;
VP= [cos(theta) -sin(theta);sin(theta) cos(theta)];
%=====
for ii=1:lrho
    rhoi=rho(ii);
    vp1=1/(1+rhoi); vp2=rhoi*vp1; % sigma_1^2 and sigma_2^2
    R=VP * diag([vp1 vp2])*VP';   % Covariance matrix
    rxy=R*gw;
    gn(:,1)=ginit;
    %==== Descent
    for n=2:N
        gn(:,n)=gn(:,n-1)+mu*(rxy-R*gn(:,n-1));
    end
    gnc(:,ii)=gn'*[1;j];
end
%=====
plot([1 j]*gw,'ro'); grid
hold on; plot(gnc); hold off
```

The results are shown in Figure 11.13. You can see that whatever the initial value, the higher the eigenvalue ratio, the slower the convergence to the Wiener solution.

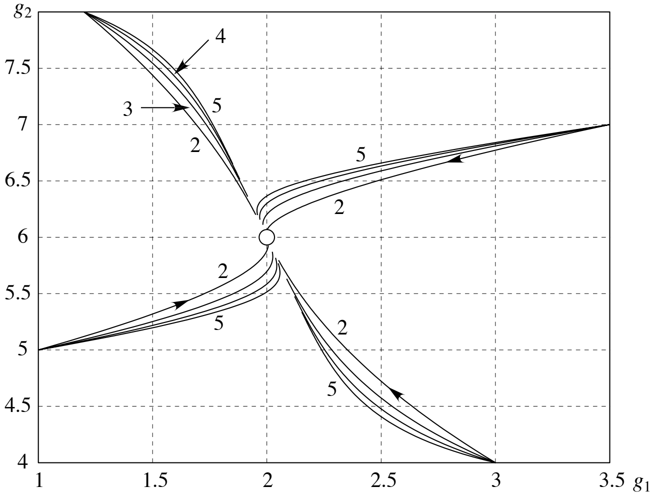


Figure 11.13 – *Gradient algorithm: evolution trajectories of the pairs $(g_0(n), g_1(n))$ as functions of the ratios $\rho \in \{2, 3, 4, 5\}$ of the two eigenvalues of the matrix* **R**, *at a constant power, for four different initialization points. The Wiener filter is* $\mathbf{g}_w = \{2,\ 6\}$

The program graddet23.m draws the ellipses associated with the criterion for each calculation point along the trajectory. The result is Figure 11.14:

```
%===== GRADDET23.M
clear; N=70;            % Number of iterations
gw=[2;6];               % Solution of the Wiener filter
ginit=[1;5];            % Initialization
mu=0.2;                 % Step
%===== Unitary matrix
theta=-3*pi/7;
VP= [cos(theta) -sin(theta);sin(theta) cos(theta)];
rho=2; vp1=1/(1+rho); vp2=rho*vp1;
R=VP * diag([vp1 vp2])*VP';
rxy=R*gw;
gn(:,1)=ginit; c=(ginit-gw)'*R*(ginit-gw);
ellipse(gw,R,c); hold on
plot(ginit(1),ginit(2),'yx')
%===== Descent
for n=2:N
    gn(:,n)=gn(:,n-1)+mu*(rxy-R*gn(:,n-1));
    c=(gn(:,n)-gw)'*R*(gn(:,n)-gw);
    ellipse(gw,R,c);
    plot(gn(1,n),gn(2,n),'yx')
end
gnc=gn'*[1;j];
plot(gnc)
%=====
plot([1 j]*gw,'yo'); grid
plot([gw(1)-VP(1,1) gw(1)+VP(1,1)],[gw(2)-VP(2,1) gw(2)+VP(2,1)])
plot([gw(1)-VP(1,2) gw(1)+VP(1,2)],[gw(2)-VP(2,2) gw(2)+VP(2,2)])
hold off
```

The vector $g(n) - g(n-1)$ can be interpreted as the vector tangent to the trajectory. But since we have:

$$g(n) - g(n-1) = \mu(\mathbf{r} - \mathbf{R}g(n-1)) = \mu\mathbf{R}(g_\infty - g(n-1)) \qquad (11.52)$$

when $n \to \infty$, the tangent vector tends to merge with the eigenvector of $\mathbf{R}$ associated with the smallest eigenvalue. This is what you can see in Figure 11.14 which shows the two directions corresponding to the eigenvectors of the matrix $\mathbf{R}$.

11.4.3 Wiener equalization

Consider the diagram in Figure 11.15. The filter $h(n)$ and the autocovariance sequences of $x(n)$ and $b(n)$ respectively are assumed to be known. $x(n)$ and $b(n)$ are also assumed to be centered and uncorrelated with each other.

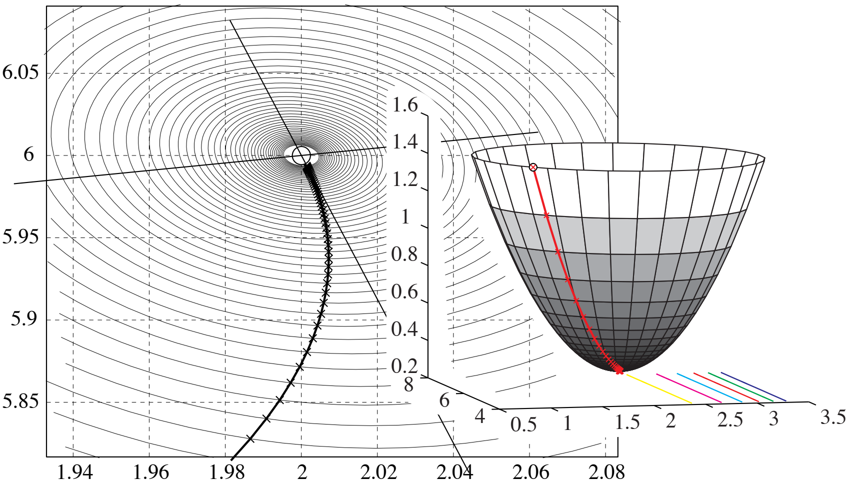


Figure 11.14 – *Gradient algorithm: trajectory with its elliptical isocriterion contours* $J(\mathrm{g}(n))$

We wish to determine the best filter $g(n)$ for finding the signal $x(n)$ based on the observation of $y(n)$. This operation is called an *equalization*. The resulting signal $\hat{x}(n)$ is not quite the same as $x(n)$ but also depends on a certain residue of terms associated with the other values of the input sequence: this residue is called *interference*.

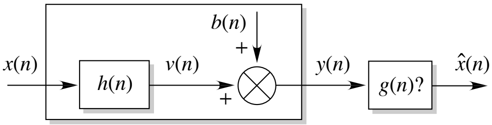


Figure 11.15 – *Equalization by Wiener filtering*

In the absence of noise, the equalizing filter has of course the impulse response $g(n)$ with $(g \star h)(n) = \delta(n)$ and the complex gain $G(f) = 1/H(f)$. Because $g(n)$ forces the interference to zero, the filter $g(n)$ is called a *zero forcing* (ZF) filter.

In the presence of noise, the zero forcing filter is not the one that minimizes the variance of the error between the sequence $x(n)$ and the equalizer output sequence $\hat{x}(n)$. The optimum filter is the Wiener filter. Without any time constraints, and based on formula 11.41 expressing the solution in the frequency domain, we have to calculate $S_{xy}(f)$ and $S_{yy}(f)$ as functions of $S_{bb}(f)$, $S_{xx}(f)$ and $H(f)$. According to the figure, and because of formulae 8.39 and 8.43 and the fact that $x(n)$ and $b(n)$ are assumed to be uncorrelated hence $S_{xb}(f) = 0$), we get:

$$S_{xy}(f) = H^*(f)S_{xx}(f) \text{ and } S_{yy}(f) = |H(f)|^2 S_{xx}(z) + S_{bb}(f)$$

Replacing these expressions in 11.41, leads us to:

$$G(f) = \frac{H^*(f)}{|H(f)|^2 + \rho(f)} \quad \text{where} \quad \rho(f) = \frac{S_{bb}(f)}{S_{xx}(f)} \tag{11.53}$$

Of course, the result is equivalent to the Zero Forcing filter $1/H(f)$ for a noise equal to zero.

As we have already said, the main problem with expression 11.53 is that the stable solution usually is not causal. This is why we will only be considering the case, which has important practical applications, where the solution to the problem is approximated by an FIR filter with a sufficient delay. We will continue to use the improper notation $g(n)$ to refer to the filter's impulse response.

The FIR filter $g(n)$ can then be obtained from expression 11.42. This means we first have to determine the expressions of $R_{yy}(k)$ and $R_{xy}(k)$ as functions of $R_{xx}(k)$, $R_{bb}(k)$ and $h(k)$. We get:

$$\begin{aligned} R_{yy}(k) &= \mathbb{E}\{y(n+k)y(n)\} = \mathbb{E}\{(v(n+k)+b(n+k))(v(n)+b(n))\} \\ &= R_{vv}(k) + R_{bb}(k) \end{aligned}$$

where $v(n)$ refers to the filter's output and where we have used the fact that $b(n)$ and $v(n)$ are not correlated. According to formula 8.42 (see page 298) which gives the autocovariance function of a linear filter's output, we have:

$$R_{vv}(k) = R_{xx}(k) \star (h(k) \star h^*(-k))$$

Meaning that:

$$R_{yy}(k) = R_{xx}(k) \star h(k) \star h^*(-k) + R_{bb}(k)$$

Likewise, we have:

$$\begin{aligned} R_{xy}(k) &= \mathbb{E}\{x(n+k)y(n)\} = \mathbb{E}\{x(n+k)(v(n)+b(n))\} \\ &= \mathbb{E}\{x(n+k)v(n)\} = R_{xv}(k) \end{aligned}$$

Formula 8.44 gives us the intercorrelation between the input and the output of a filter. Using it leads to:

$$R_{xv}(k) = h(-k) \star R_{xx}(k)$$

To sum up, the expressions of $R_{yy}(k)$ and $R_{xy}(k)$ as functions of $h(k)$, $R_{xx}(k)$ and $R_{bb}(k)$ are:

$$\begin{aligned} R_{yy}(k) &= R_{xx}(k) \star h(k) \star h^*(-k) + R_{bb}(k) \\ R_{xy}(k) &= h(-k) \star R_{xx}(k) \end{aligned}$$

At this point, we can construct the matrix $\mathbf{R}$ and the vector $\mathbf{r}$ and use formula 11.43 to find the coefficients of the filter $g(n)$.

Let us assume, for example, that $h(k) = 0$ for $k \notin \{0, \dots, L-1\}$, that the signal $x(n)$ is white and has the power σ_x^2, and that the noise is white, with the power σ_b^2. This means that $R_{yy}(k) = \sigma_x^2 h(k) \star h^*(-k) + \sigma_b^2 \delta(k)$ and $R_{xy}(k) = \sigma_x^2 h(-k)$. We can then determine the expressions of $\mathbf{R}$ and $\mathbf{r}$ involved in 11.42 and 11.43:

$$\begin{cases} \mathbf{R} = \sigma_x^2 \mathbf{C}_h + \sigma_b^2 \mathbf{I} \\ \mathbf{r} = \sigma_x^2 \mathbf{h} \end{cases}$$

where $\mathbf{C}_h$ is an $N \times N$ Toeplitz matrix constructed from the sequence $h(n) \star h^*(-n)$ and where $\mathbf{h}$ is:

$$\mathbf{h}^T = \left[\underbrace{0 \;\; \dots \;\; 0}_{A} \;\; \underbrace{h(0) \;\; \dots \;\; h(L-1) \;\; 0 \;\; \dots \;\; 0}_{C} \right]$$

We will come back to the concept of equalization when we discuss the important case of communications systems (paragraph 12.17) with a sequence of symbols as the input.

11.5 The LMS (least mean square) algorithm

11.5.1 The constant step algorithm

We now come back to equation 11.42, which gives us the Wiener filter. In theory, if we perfectly knew the autocovariance matrix $\mathbf{R}$ and the intercovariance vector $\mathbf{r}$, equation 11.42 would give us the solution to the problem in the form $\mathrm{g} = \mathbf{R}^{-1}\mathbf{r}$. The solution could then be obtained numerically, using the gradient algorithm seen previously and defined by the recursive equation 11.46 rewritten here:

$$\mathrm{g}(n) = \mathrm{g}(n-1) + \mu \left(\mathbf{r} - \mathbf{R}\mathrm{g}(n-1) \right) \tag{11.54}$$

For many practical problems, the autocovariance matrix $\mathbf{R}$ and the covariance vector $\mathbf{r}$ are not known, and have to be estimated from the data observed. Also, the stationarity hypotheses are never completely verified in the long run, meaning a single estimation at the beginning of the process is not enough. These quantities have to be re-estimated regularly. The LMS algorithm returns a simple adaptive solution to the problem.

Let $\mathbf{y}(n) = [y(n) \quad y(n-1) \quad \ldots \quad y(n-P+1)]^T$ be the vector constructed from the last P observations, and let:

$$\begin{aligned} e(n) &= x(n) - \hat{x}(n) \\ &= x(n) - \begin{bmatrix} y(n) & y(n-1) & \ldots & y(n-P+1) \end{bmatrix} \begin{bmatrix} g_{n-1}(0) \\ g_{n-1}(1) \\ \vdots \\ g_{n-1}(P-1) \end{bmatrix} \\ &= x(n) - \mathbf{y}^T(n)\mathbf{g}(n-1) \end{aligned} \tag{11.55}$$

be the difference between the value $x(n)$ observed at the time n and the value $\hat{x}(n)$ calculated from the observation of $\mathbf{y}(n)$, with the use of the coefficients $\mathbf{g}(n-1)$ obtained at the *previous* time. Be aware that $e(n)$ is different from the difference found in expression 11.39 and involving the desired Wiener filter $\mathbf{g}$. With these notations, we have at the $(n-1)$-th step:

$$\mathbf{r} - \mathbf{R}\mathbf{g}(n-1) = \mathbb{E}\{x(n)\mathbf{y}(n)\} - \mathbb{E}\left\{\mathbf{y}(n)\mathbf{y}(n)^T\right\}\mathbf{g}(n-1) = \mathbb{E}\{\mathbf{y}(n)e(n)\}$$

If we replace the previous expression in 11.54, we get:

$$\mathbf{g}(n) = \mathbf{g}(n-1) + \mu\mathbb{E}\{\mathbf{y}(n)e(n)\} \tag{11.56}$$

An idea we owe to *Widrow* [102] is to replace $\mathbb{E}\{\mathbf{y}(n)e(n)\}$ in equation 11.56 with its "instantaneous" value $\mathbf{y}(n)e(n)$, leading to the following two equations that make up the *stochastic gradient* algorithm, also called simply the *LMS algorithm*:

Initial value: $\mathbf{g}(0) = \mathbf{0}$,
Repeat:

$$\left| \begin{aligned} e(n) &= x(n) - \mathbf{y}^T(n)\mathbf{g}(n-1) \\ \mathbf{g}(n) &= \mathbf{g}(n-1) + \mu\mathbf{y}(n)e(n) \end{aligned} \right. \tag{11.57}$$

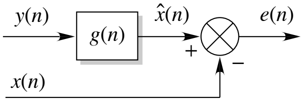


Figure 11.16 – *A reproduction of Figure 11.10*

You can see in Figure 11.16 that the algorithm can be interpreted in a simple way: if $e(n)$ is null, then we can reasonably consider that the coefficients have the right value, in which case they are left unchanged. Otherwise, the correction made to $\mathbf{g}(n-1)$ by the term $\mu\mathbf{y}(n)e(n)$ increases with $e(n)$.

The idea of replacing the mathematical expectations with the instantaneous values is sometimes associated with other criteria, such as the least squares criterion, and leads to uneven results. As we did with expression 11.44, we start with the expression $J(\mathbf{g})$ of the criterion to minimize with respect to $\mathbf{g}$, then we determine the expression of its gradient with respect to g, and finally, to obtain a minimum, we use the recursive equation:

$$\mathbf{g}(n) = \mathbf{g}(n-1) - \frac{\mu}{2}\left.\frac{\partial J(\mathbf{g})}{\partial \mathbf{g}}\right|_{\mathbf{g}=\mathbf{g}(n-1)} \tag{11.58}$$

If there are now unknown mathematical expectations in expression 11.58, we can replace these expectations with instantaneous quantities obtained by the plain and simple subtraction of the mathematical expectation, the same way we did with the LMS algorithm. This leads us to a *stochastic gradient* type algorithm. However, the results do not always meet expectations, whereas in the case of the quadratic criterion, the results are quite good, as we are going to see in some examples.

But in any case, we have to ask ourselves the following questions:

– Are there initial values $\mathbf{g}(0)$ and values of μ that ensure convergence (the exact definition will have to be given later on) since $\mathbf{g}(n)$ is random?

– If there is convergence, does $\mathbf{g}(n)$ lead to the global minimum of $J(\mathbf{g})$ or to some possible unwanted local minima?

– In the case of the existence of several minima, is there a practical condition on the choice of the initial value that ensures convergence to the global minimum?

These are difficult problems to solve because removing the mathematical expectation from the recursive equation makes the analysis very complex. A certain number of answers can be found in the literature, but they use hypotheses that usually are not well verified in practice. An in-depth approach can be found, for example, in [44], [60].

In practice, the following procedure is often used for setting the value of μ: μ is progressively increased until the algorithm diverges, then decreased by at least 10%. Once the value of μ is set, the results are presented by displaying the evolution, in dB, of the square of the instantaneous square deviation $p(n) = e^2(n)$. However, because the shape of $p(n)$ is often quite chaotic, it is a good idea to smooth $p(n)$ by calculating the mean $q(n)$ of N consecutive values. This can be done by using the `qn=filter(h,1,pn)` function with `h=ones(N,1)/N`. The following expression can also be used:

$$q(n) = (1-\alpha)q(n-1) + \alpha p(n)$$

where α is a forget factor that can be chosen equal to 0.1. The closer α gets to 0, the smoother the fluctuations of $p(n)$ become.

Unlike in the deterministic situation, there appears, on average a misadjustment at the convergence (see definition 11.48) causing oscillations around the minimum. What should be remembered is that the misadjustment decreases when μ decreases and when the number of coefficients of g decreases. As μ gets smaller, the rise time decreases and the tracking capability declines. Therefore we must find a compromise that takes these two requirements into account. The following examples as well as paragraph 11.5.3 deal with these behaviors.

Example 11.3 (Suppression of a single tone jammer)
Consider a signal $s(n)$ corrupted by a single tone jammer. The observed signal can be written $x(n) = s(n) + b(n)$ where $b(n) = A\cos(2\pi f_b n + \phi)$ refers to the jammer, the frequency f_b of which is assumed to be known, while its amplitude A and its phase ϕ are unknown. $b(n)$ can also be expressed as $b(n) = g_c \cos(2\pi f_b n) + g_s \sin(2\pi f_b n)$ where g_c and g_s are the unknown parameters we have to estimate.

The problem of extracting $b(n)$ from the observed signal $x(n)$ is identical in every way to the one we encountered in example 8.6 on suppressing a trend. The solution suggested here uses the LMS algorithm, hence it has the advantage of being adaptive.

1. With the help of the diagram on Figure 11.17, prove that the problem is equivalent to finding a two input filter.

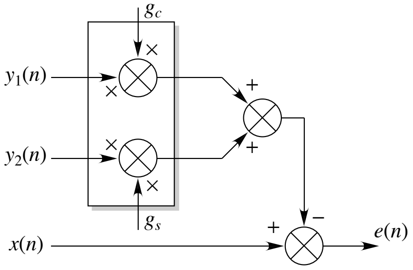


Figure 11.17 – *Two input Wiener filter*

Determine the Wiener solution of this filter that minimizes the least squares criterion:

$$J(\mathrm{g}) = \sum_n (x(n) - [g_c \cos(2\pi f_b n) + g_s \sin(2\pi f_b n)])^2$$

Use this result to find the adaptation equations of the associated LMS algorithm.

2. In order to test the tracking capability (adaptivity) of the LMS, we now assume that the jammer is amplitude modulated according to the expression:

$$b(n) = A(1 + k\cos(2\pi f_m n))\cos(2\pi f_b n) \tag{11.59}$$

where the amplitude A, the constant k and the frequency f_m are unknown. We assume as our hypothesis that this signal can be approximated by a sequence of time translated sines of the type:

$$b(n) \simeq \sum_{k=1}^{P} [g_c(k)\cos(2\pi f_b(n-k)) + g_s(k)\sin(2\pi f_b(n-k))]$$

where the sequences $\{g_c(k)\}$ and $\{g_s(k)\}$ are unknown quantities we are going to obtain by minimizing the square deviation between $x(n)$ and $b(n)$. Show that the problem is also equivalent to finding a two input filter. Give this filter's impulse responses. Write a program, using the LMS algorithm, that eliminates such a jammer for a speech signal.

3. Write a program that generates a $f_b = 1{,}000$ Hz jammer modulated at $f_m = 20$ Hz according to expression 11.59. Let $A = 1$ and $k = 0.5$. After applying this jammer to a speech signal. Implement the LMS algorithm to try and eliminate it. Choose P empirically by listening to the result.

HINT:

1. The signal observed is $x(n) = s(n) + b(n)$ where the expression of the jammer $b(n)$ is:

$$b(n) = g_c \cos(2\pi f_b n) + g_s \sin(2\pi f_b n) \tag{11.60}$$

Equation 11.60 can be interpreted as a FIR filtering of two input signals $y_1(n) = \cos(2\pi f_b n)$ and $y_2(n) = \sin(2\pi f_b n)$. The coefficients of the filter are g_c and g_s respectively. If you refer to Figure 11.16, you will see that the two-dimension signal $(\cos(2\pi f_b n), \sin(2\pi f_b n))$ represents the signal $y(n)$ shown in the figure, and the (two-dimension) filter's two coefficients

g_c and g_s. As for the signal $e(n)$, it represents the denoised signal. This is shown in Figure 11.17. We can then determine the two coefficients g_c and g_s that minimize the square deviation between $b(n)$ and $x(n)$. $s(n)$ therefore behaves as the unwanted signal when estimating the two quantities. The LMS algorithm, which estimates g_c and g_s, is described by the two equations:

$$\begin{cases} e(n) = x(n) - [\cos(2\pi f_b n) \quad \sin(2\pi f_b n)] \begin{bmatrix} g_c(n-1) \\ g_s(n-1) \end{bmatrix} \\ \begin{bmatrix} g_c(n) \\ g_s(n) \end{bmatrix} = \begin{bmatrix} g_c(n-1) \\ g_s(n-1) \end{bmatrix} + \mu e(n) \begin{bmatrix} \cos(2\pi f_b n) \\ \sin(2\pi f_b n) \end{bmatrix} \end{cases}$$

2. To take into account the modulation phenomenon, or other aspects of the jammer, let us assume that the jammer's expression is a sum of P delayed sines of the kind:

$$b(n) = \sum_{k=1}^{P} [g_c(k)\cos(2\pi f_b(n-k)) + g_s(k)\sin(2\pi f_b(n-k))]$$

This expression can be seen as the sum of the filtering of $y_1(n) = \cos(2\pi f_b n)$ by the filter $g_c(n)$ and of the filtering of $y_2(n) = \cos(2\pi f_b n)$ by the filter $g_s(n)$. If we choose to minimize the least squares criterion $J(\mathbf{g})$, then the LMS algorithm is associated with:

$$\begin{cases} e(n) = x(n) - \mathbf{g}^T(n-1) \begin{bmatrix} \mathbf{C}(n) \\ \mathbf{S}(n) \end{bmatrix} \\ \mathbf{g}(n) = \mathbf{g}(n-1) + \mu e(n) \begin{bmatrix} \mathbf{C}(n) \\ \mathbf{S}(n) \end{bmatrix} \end{cases}$$

where:

$$\begin{aligned} \mathbf{g}(n) &= \big[[g_c(0) \quad \ldots \quad g_c(P-1)] \quad [g_s(0) \quad \ldots \quad g_s(P-1)o]\big]^T \\ \mathbf{C}(n) &= \big[\cos(2\pi f_b n) \quad \ldots \quad \cos(2\pi f_b(n-P+1))\big]^T \\ \mathbf{S}(n) &= \big[\sin(2\pi f_b n) \quad \ldots \quad \sin(2\pi f_b(n-P+1))\big]^T \end{aligned}$$

The signal $e(n)$ represents the denoised signal.

3. Load the previously recorded speech signal s and execute:

```
%===== scrambexple.m
load phrase
N=length(sn); sn=sn/max(abs(sn));
```

```
%===== Modulation (A=1 for a pure jammer)
Fb=1000; Fe=8000; fb=Fb/Fe; Fm=20; fm=Fm/Fe ;
A=1; k=0.5;
b=A*(1+k*cos(2*pi*fm*(1:N)')) .* cos(2*pi*fb*(1:N)');
%===== Jammed signal
x=sn+b;
%===== LMS implementation
mu=0.01; P=30; gch=zeros(2*P,1);
%===== schap is the reconstructed signal
schap=zeros(N,1);
for n=P:N
    gY=[cos(2*pi*fb*(n:-1:n-P+1)'); sin(2*pi*fb*(n:-1:n-P+1)')];
    en0=x(n)-gch'*gY; gch=gch+mu*en0*gY;
    schap(n)=en0;
end
subplot(311); plot(sn); grid; subplot(312); plot(x); grid
subplot(313); plot(schap); set(gca,'ylim',[-1 1]); grid
```

■

Example 11.4 (Linear prediction)
Prove that the problem of the linear prediction coefficient calculation can be solved by an LMS algorithm. Apply this result to the estimation of an AR-2 process's coefficients. Plot the evolution of the square deviation for several trials by choosing values of μ that ensure "convergence" to a small and relatively constant value.

Hint: linear prediction consists of estimating the value at the time n based on the last P values, according to the expression:

$$\hat{x}(n) = g_0 x(n-1) + \cdots + g_{P-1} x(n-P)$$

The diagram in Figure 11.16 shows that the sequence $x(n-1), \ldots, x(n-P)$ acts as the observation $y(n)$, whereas $x(n)$ acts as the reference signal $x(n)$ in the diagram. This means we have to replace the vector $\mathbf{y}(n)$ by $\mathbf{x}(n-1) = [x(n-1) \quad \ldots \quad x(n-P)]^T$ in the equations 11.57 of the LMS algorithm. We get:

$$\left\{\begin{array}{l} e(n) = x(n) - \begin{bmatrix} g_0(n-1) & \cdots & g_{P-1}(n-1) \end{bmatrix} \begin{bmatrix} x(n-1) \\ \vdots \\ x(n-P) \end{bmatrix} \\ \begin{bmatrix} g_0(n) \\ \vdots \\ g_{P-1}(n) \end{bmatrix} = \begin{bmatrix} g_0(n-1) \\ \vdots \\ g_{P-1}(n-1) \end{bmatrix} + \mu e(n) \begin{bmatrix} x(n-1) \\ \vdots \\ x(n-P) \end{bmatrix} \end{array}\right.$$

Notice that the Wiener solution provided by the equation $\mathbf{Rg} = \mathbf{r}$, an equation where $\mathbf{r}$ is the vector with the following P components:

$$r(k) = \mathbb{E}\{x(n)x(n-k)\}\,, k \in (1, \ldots, P)$$

and $\mathbf{R}$ is the matrix with the generating element:

$$\mathbb{E}\{x(n)x(n-k)\}\,, k \in (0, \ldots, P-1)$$

are simply the Yule-Walker equations 8.58. We saw on page 411 that, for an AR process with its poles inside the unit circle, the linear prediction coefficients coincide with the model's coefficients.

The `estARlms.m` program generates an AR-2 and estimates its coefficients using the LMS algorithm:

```
%===== ESTARLMS.M
%===== Generation of the AR-2 signal
a=[1 -1.6 0.8]'; P=length(a)-1;
N=4000; w=randn(N,1);      % sigma2w=1
x=filter(1,a,w);           % Signal
%===== Estimation of the eigenvalues
r0=x' * x/N; r1=x(2:N)'*x(1:N-1)/N;
r2=x(3:N)'*x(1:N-2)/N;     % or D2=[x' 0; 0 x'];
Rx=toeplitz([r0 r1]);      % and Rx=D2*D2'/N;
lambda=eig(Rx); muMax=2/max(lambda) % Estimated max.
%===== LMS implementation
mu=0.005; gest=zeros(P,1); err=[];
for n=P+1:N
    y=x(n-1:-1:n-P); en0=x(n)-gest'*y;
    err=[err; en0]; gest=gest + mu*en0*y;
end
aest=[1;-gest];
%===== Yule-Walker equation
aYule=[1;-inv(Rx) * [r1;r2]];
%===== Displaying the results
[a aest aYule], Jlim=std(err)^2
sigma2w_estime=r0 + aYule(2)*r1 + aYule(3)*r2
```

The program estimates the eigenvalues of the covariance matrix to find the maximum value of $\mu_d = 2/(\max_i \lambda_i)$ that ensures the convergence of the deterministic gradient algorithm. If we now compare the convergences of the stochastic gradient algorithm, we notice that μ_d is much greater than μ. At the convergence, the estimation slightly oscillates around the solution to the equation $\mathbf{Rg} = \mathbf{r}$. As we have said, the fact that we eliminated the expectations in the gradient algorithm introduces a misadjustment. The program gives the value of the instantaneous square deviation $e(n)$ (variable `err`) as well as its variance denoted by `Jlim` in the program. This value should be compared with the theoretical value $J_{\min}$, which in this case is equal to $\sigma_w^2 = 1$. ■

Exercise 11.4 (LMS algorithm: channel identification)
Consider the channel identification problem. A training signal $x(n)$ is sent, producing the output $v(n) = h_0(n) \star x(n)$. Let us assume that the unknown filter $h_0(n)$ is an FIR filter with P coefficients. In the presence of noise, the observation $y(n)$ has the expression $y(n) = v(n) + b(n)$ where the noise $b(n)$ is assumed to be white.

In practice, when the length of the filter $h_0(n)$ is unknown, a large value can be chosen in order to be sure that it is overestimated. In this case, we will consider that this length is known, and we are going to determine a P coefficient filter such that $\hat{y}(n) = (h \star x)(n)$ is the one that best looks like $y(n)$ (see Figure 11.18).

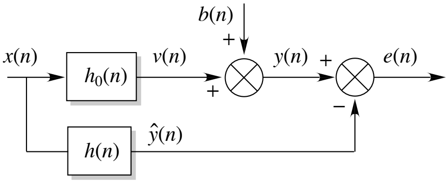


Figure 11.18 – *Identification of an h_0 channel*

In mathematical terms, we have to find the $h(n)$ that minimizes the square deviation between $y(n)$ and:

$$\hat{y}(n) = h(0)x(n) + \cdots + h(P-1)x(n-P+1)$$

Minimizing the square deviation leads to the Wiener solution given by $\mathbf{Rg} = \mathbf{r}$, where $\mathbf{r}$ is the vector whose P components have the expressions $\mathbb{E}\{y(n)x(n-k)\}$ for k from 0 to $P-1$ and $\mathbf{R}$ is the matrix with the generating term $\mathbb{E}\{x(n)x(n-k)\}$ for k from 0 to $P-1$.

We choose to use the LMS algorithm to estimate $h_0(n)$, rather than to solve this equation. Write a program that simultaneously performs:

- Input and output signal simulation: the input signal $x(n)$ is a white noise with a power equal to 1. The output signal is $y(n) = h(n) \star x(n) + b(n)$. $h(n)$ is the finite impulse response that simulates the channel. Let `h=[1 0.6 0.3]'`.

  $b(n)$ is a white noise. Determine its variance based on the value of the signal-to-noise ratio between $b(n)$ and $x(n)$, expressed in dB.

- Implementation of the LMS algorithm to identify $h_0(n)$.

- Presentation of the results: display the value in dB of the estimation error's power. Because this sequence of values is very chaotic, smooth it by calculating a mean over a hundred values, using the `filter` function or a forget factor.

Study the convergence speed and the misadjustement for various values of μ. What happens if μ is too high? Compare with the value $\mu = 2/\max_i \lambda_i$.

11.5.2 The normalized LMS algorithm

As a reminder, here is the structure of the LMS algorithm in its standard form:

$$\mathbf{g}(n) = \mathbf{g}(n-1) + \mu \mathbf{y}(n) \left[x(n) - \mathbf{y}^T(n)\mathbf{g}(n-1)\right]$$

This version of the algorithm is not well adapted to the observation signal's power variations. This is because, as we have said, μ has to be chosen according to the inverse of the highest eigenvalues of $\mathbf{R}$. Remember that the sum of these eigenvalues precisely represents the power of $y(n)$. Hence, in the practical situation of signals that are not stationary and a power that varies, we have to adjust μ as time goes by. If we do not, the algorithm might diverge. Hence the idea of taking as the step's expression:

$$\mu(n) = \frac{\lambda}{P_y(n)}$$

The step is then said to be *normalized.* We still have to find a simple way of estimating $P_y(n)$ as time goes by. In order to do so, we can use one of the following simple expressions:

- $P_y(n) = \mathbf{y}^T(n)\mathbf{y}(n)/P$ where P is the length of the filter we wish to estimate,
- or the empirical estimator:

$$P_y(n) = \frac{1}{n}(y^2(1) + y^2(2) + \cdots + y^2(n)) = (1 - \frac{1}{n})P_y(n-1) + \frac{1}{n}\,y^2(n)$$

- or a recursive expression with a forget factor $0 < \alpha < 1$ of the type:

$$P_y(n) = (1-\alpha)P_y(n-1) + \alpha y_n^2 \tag{11.61}$$

This leads to what is called the *normalized LMS* algorithm:

```
initial value: $\mathbf{g}(0) = \mathbf{0}$, Repeat:
    $e(n) = x(n) - \mathbf{y}^T(n)\mathbf{g}(n-1)$
    $P_y(n) = (1-\alpha)P_y(n-1) + \alpha y_n^2$
    $\mu(n) = \frac{\lambda}{P_y(n) + \varepsilon}$
    $\mathbf{g}(n) = \mathbf{g}(n-1) + \mu(n)\mathbf{y}(n)e(n)$
```

The positive quantity ε is only used to prevent $\mu(n)$ from becoming too high, particularly in the case of a rather long drop in the power of $y(n)$.

You can do exercise 11.4 over again replacing the standard LMS algorithm with the normalized one. You should focus on the cases of sharp power transitions to compare the tracking capabilities of the two algorithms. The normalized LMS often comes out with the better results.

The following program compares how the standard and normalized LMS algorithms perform in an equalization program. This is done by generating a signal with a power that varies, and then to consecutively filter it with two different filters. The equalization is performed with a slightly too long FIR filter, since it has a length of $P = 6$ coefficients. The values of `muS` and `muN` are set by trial and error, choosing the α that ensured stability:

```
%===== COMPLMS.M
%===== Signal generation
clear; N=1000; x0=randn(1,N); x1=.2*randn(1,N);
x=[x0';x1';x0']; subplot(411); plot(x); grid
%===== Filtering and addition of noise
h0=[1 0.7]; v0=filter(h0,1,x0);
h1=[1 0.3]; v1=filter(h1,1,x1);
v=[v0';v1';v0']; N=length(v);
SNR=30; Px=x'*x/N; sigma=sqrt(Px)*10^(-SNR/20);
b=sigma*randn(N,1); y=v+b;
subplot(412); plot(y); grid
%===== Equalization using a P coefficient FIR filter
P=6;
HestS=zeros(P,1);    % Standard LMS
HestN=zeros(P,1);    % Normalized LMS
enS=zeros(N-P+1,1);  % Error for the standard LMS
enN=zeros(N-P+1,1);  % Error for the normalized LMS
%===== Implementing the LMSs
muS=0.06;            % Standard LMS step
muN=0.08;            % Normalized LMS step
%===== Forget factor
alpha=0.05; umalpha=1-alpha;
%==== Py(n) initial
pyn=y(1:P-1)'*y(1:P-1)/P;
%===== Algorithms
for n=P:N
    %===== Standard
    en0S=x(n)-HestS'*y(n:-1:n-P+1);
    HestS=HestS+muS*en0S*y(n:-1:n-P+1); enS(n-P+1)=en0S;
    %===== Normalized
    en0N=x(n)-HestN'*y(n:-1:n-P+1);
    % Two expressions for the estimation of Py
    pyn=y(n:-1:n-P+1)'*y(n:-1:n-P+1)/P;
    %pyn=umalpha*pyn+alpha*y(n)*y(n);
    HestN=HestN+muN*en0N*y(n:-1:n-P+1)/pyn;
    enN(n-P+1)=en0N;
end
%===== Displaying the results
en2N=enN .^2; en2S=enS .^2;
moy=100; hmoy=ones(1,moy)/moy;
en2moyS=filter(hmoy,1,en2S(1:N-P+1));
en2moyN=filter(hmoy,1,en2N(1:N-P+1));
endBS=10 * log10(en2moyS(moy:N-P+1));
endBN=10 * log10(en2moyN(moy:N-P+1));
subplot(212); plot(endBS); grid
hold on; plot(endBN,'r'); hold off
```

The results are shown in Figure 11.19. You can see that when there is a loss of stationarity, the standardized LMS algorithm has the better response. In the stationary parts, however, it behaves, in this example, basically in the same way as the standard LMS algorithm. You can try this with several values of the signal-to-noise ratio, or use a signal speech instead of the signal `x`.

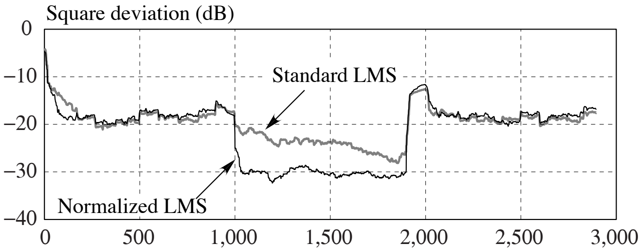


Figure 11.19 – *Comparing the tracking capabilities of the standard LMS and the normalized LMS*

Comments

The standard and normalized LMS algorithms are different from any other adaptive algorithm because of how easy they are to implement. The gradient step is the only parameter that has to be set. This setting often requires a number of trials to ensure both the convergence of the algorithm (small value of μ) and a good enough tracking capability (large value of μ).

Performances greatly depend on the covariance matrix of the observation, and more particularly on how far apart the eigenvalues are from each other. The highest eigenvalue is related to the final error obtained at the convergence and the lowest eigenvalue is related to the total convergence time of the algorithm.

Notice, finally, that one of the hypotheses assumed for the determination of the Wiener filter states that the channel is time-invariant. Yet, what is expected of the adaptive algorithm is not so much to reach a stationary solution that may not exist so much as to follow a system that varies slowly. In this case, the LMS is an acceptable solution.

11.5.3 Echo canceling

Echo canceling is an important example of the practical use of the LMS algorithm.

In some situations such as with a "handsfree" device on a cellular phone, a speaker is located close to a microphone (Figure 11.20). The signal emitted by the speaker is transmitted along an acoustic path, dependent on where the user is, and reaches the microphone, creating an unwanted echo. In practice, a filter with an unknown impulse response, sometimes with hundreds of coefficients, can be used as a model to describe the acoustic path.

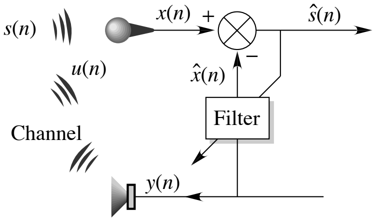


Figure 11.20 – *The echo cancelation principle*

Let $y(n)$ be the speaker's output signal, $u(n)$ the resulting echo in front of the microphone, $s(n)$ the microphone input signal we wish to send, and $x(n)$ the microphone output signal observed. We can write:

$$x(n) = s(n) + u(n) = s(n) + h(n) \star y(n)$$

where $h(n)$ is the impulse response that serves as the model for the acoustic path from the speaker's output to the microphone's input. The difficulty in echo cancellation is to estimate $s(n)$ based on two observed sequences $x(n)$ and $y(n)$. This is done by assuming that:

- the sequences $s(n)$ and $y(n)$ are centered and uncorrelated;
- the estimation of $s(n)$ is linear, of the type $\hat{s}(n) = x(n) - \mathbf{g}^T\mathbf{y}(n)$, where the filter $\mathbf{g}$, the length of which is P, is determined by minimizing:

$$K(\mathbf{g}) = \mathbb{E}\left\{|s(n) - \hat{s}(n)|^2\right\}$$

We are going to prove that, in this case, echo cancellation is equivalent to determining the Wiener filter according to the diagram in Figure 11.10. We have:

$$\begin{aligned}K(\mathbf{g}) &= \mathbb{E}\left\{|s(n)-\hat{s}(n)|^2\right\} = \mathbb{E}\left\{(|s(n)-(x(n)-\mathbf{g}^T\mathbf{y}(n))|^2\right\}\\ &= \mathbb{E}\left\{|s(n)|^2\right\} - 2\mathbb{E}\{s(n)x(n)\} - \\ &\qquad 2\mathbb{E}\left\{s(n)\mathbf{y}^T(n)\right\}\mathbf{g} + \mathbb{E}\left\{\left|x(n)-\mathbf{g}^T\mathbf{y}(n)\right|^2\right\}\end{aligned}$$

where the second term is null because $s(n)$ and $x(n)$ are uncorrelated and centered. Hence, the minimization of $K(\mathbf{g})$ with respect to $\mathbf{g}$ is equivalent to that of:

$$J(\mathbf{g}) = \mathbb{E}\left\{\left|x(n)-\mathbf{g}^T\mathbf{y}(n)\right|^2\right\}$$

which does not involve $s(n)$. Therefore, the LMS algorithm can be used to estimate $\mathbf{g}$ based on the sequences $x(n)$ and $y(n)$, then to substract the signal $\mathbf{g}^T\mathbf{y}(n)$ from $x(n)$ to obtain an estimate of the signal $s(n)$.

Absence of vocal activity

We are first going to consider the situation where there is no vocal activity, that is to say when $s(n) = 0$ and perform the following simulation using MATLAB®: the signal $y(n)$ is a white noise and the signal that represents $x(n)$ is created by filtering $y(n)$ with the finite impulse response filter $\{1 \quad 0.3 \quad -0.1 \quad 0.2\}$. Let $y(n)$ be the filtered signal. The following program implements the LMS algorithm:

```
%===== ECHOCANCEL1.M
clear; N=4000; alpha=0.2;
yn=randn(N,1);      %=== Reference
hh=[1 0.3 -0.1 0.2];
xn=filter(hh,1,yn); %=== Echo
%===== LMS implementation
mu=0.05; P=20; gn=zeros(P,1);
en=zeros(N,1);
for n=P:N
    en0=xn(n)-gn'*yn(n:-1:n-P+1);
    gn=gn+mu*en0*yn(n:-1:n-P+1);
    en(n)=(1-alpha)*en(n-1)+alpha*abs(en0)^2;
end
%===== Displaying the results
plot(10*log10(en(P+1:N)));
grid; set(gca,'xlim',[0 3000])
%=====
%plot(20*log10(abs(fft(hest,1024))))
```

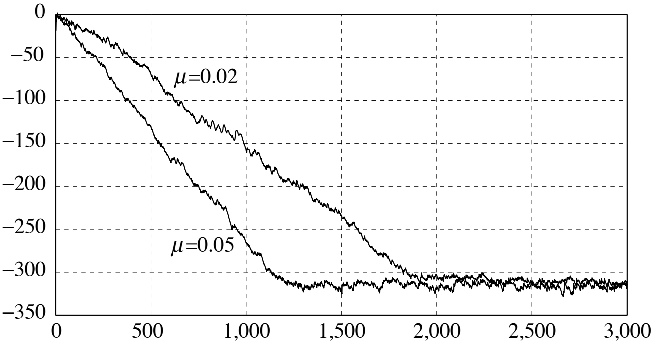


Figure 11.21 – *LMS algorithm: evolution of the square deviation, in dB, as a function of the number of steps of the algorithm, for the two values* $\mu = 0.02$ *and* $\mu = 0.05$ *of the step. The echo is a white noise filtered by the filter with the finite impulse response* $\{1; 0.3; -0.1; 0.2\}$

Figure 11.21 shows the values in decibels of the error squared, integrated with a forget factor equal to 0.2, plotted against the number of steps of the algorithm. The values $\mu = 0.02$ and $\mu = 0.05$ lead to convergence.

The results are particularly good: because the signal $s(n)$ is null, the output signal is also null, or at least should be if the calculations were completely accurate. Notice that if the signal $y(n)$ is no longer stationary or if the acoustic transfer varies with time, it would be better to use the normalized step LMS algorithm.

Presence of vocal activity

In the situation where there is vocal activity in front of the microphone, the signal $s(n)$ created by the user of the microphone behaves as a signal added to the signal $x(n)$. This makes it more difficult to adapt the algorithm, especially given the fact that $s(n)$ is relatively powerful compared to the signal $x(n)$. The following program conducts trials with the signal $s(n)$ as the speech signal. The echo signal $u(n)$ is still a filtered white noise. In this case, the error signal, when the echo is perfectly cancelled, should be a rather accurate copy of the signal that entered the microphone. Figure 11.21 shows the results: the cancellation is satisfactory after about 200 samples:

```
%===== ECHOCANCEL2.M
clear; load phrase    %=== Signal (sn)
sn=sn(:); N=length(sn); mm=max(abs(sn));
sn=sn/mm;
yn=randn(N,1);        %=== Reference
hh=[1 0.3 -0.1 0.2]; echo=filter(hh,1,yn); xn=sn+echo;
subplot(311); plot(sn); grid; set(gca,'xlim',[3500 6800])
subplot(312); plot(xn); grid; set(gca,'xlim',[3500 6800])
%===== Implementing the LMS
mu=0.01; P=20; gn=zeros(P,1);
en=zeros(N,1);        %=== Denoised signal
for n=P:N
    en0=xn(n)-gn'*yn(n:-1:n-P+1);
    gn=gn+mu*en0*yn(n:-1:n-P+1);
    en(n)=en0;
end
%===== Displaying the results
subplot(313); plot(en); grid
set(gca,'xlim',[3500 6800])
%===== Audio tests
%soundsc(sn,8000) %=== Original signal
%soundsc(xn,8000) %=== Signal with echo
%sound(en,8000) %=== Signal after echo cancelling
```

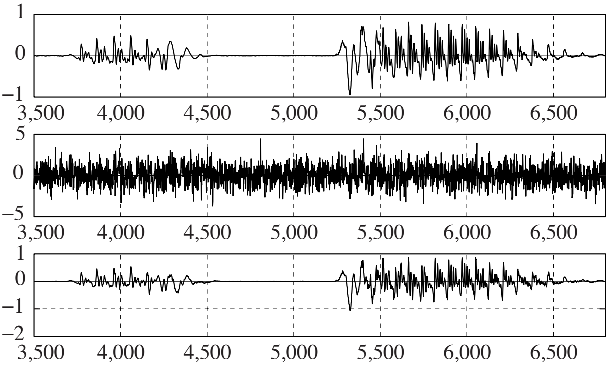


Figure 11.22 – *LMS algorithm: top graph: speech signal without the echo; middle graph: speech signal after adding the echo, which is a white noise filtered by the filter with the finite impulse response* $\{1; 0.3; -0.1; 0.2\}$*; bottom graph: processed signal*

The program can also be used to test the tracking capability by varying either the filter or the power of the signal $y(n)$.

Double talking

Finally, we can test the LMS algorithm by choosing speech signals for $s(n)$ and $y(n)$. In this case, adapting the algorithm is more difficult, because the signal $y(n)$ is no longer stationary and has spectral properties that are close to those of $s(n)$. It is then better to use both the normalized LMS algorithm and a vocal activity detector in front of the microphone. When the latter detects a signal from the user, the adaptation of the algorithm is blocked, and the echo keeps on being cancelled with the coefficients used until then. After making a few settings, the algorithm provides results that are quite satisfactory.

11.6 Application: the Kalman algorithm

In this paragraph, we will use the following notations:

- $(s|\mathcal{C})$ refers to the orthogonal projection of s onto $\mathcal{C}$;
- (x, y) refers to the scalar product of x and y, knowing that in the real deterministic case, it is denoted by $x^T y$, and in the random case by $\mathbb{E}\{xy\}$;
- $(x|y_1, \cdots, y_n)$ refers to the orthogonal projection of x onto the space generated by $(y_1, \cdots, y_n)$.

Let x and y be any two elements of a Hilbert space $\mathcal{H}$. By assuming $P = 1$ in the formula 11.3, we get, for the orthogonal projection of x onto y, expression 11.62 (the projection is denoted by αy with α such that $(y, (x - \alpha y)) = 0$, hence the result).

$$(x|y) = \frac{(y, x)}{(y, y)}\, y \tag{11.62}$$

Consider a space $\mathcal{H}$, $\mathcal{C}$ a subspace of $\mathcal{H}$, and s a vector of $\mathcal{H}$ orthogonal to all the elements of $\mathcal{C}$. Let $\mathcal{D}$ be the subspace generated by $\mathcal{C}$ and s. Then for any element of $\mathcal{H}$, we have:

$$(x|\mathcal{D}) = (x|\mathcal{C}) + (x|s) \tag{11.63}$$

Let $\{y_n\}$ be a sequence of vectors of a space $\mathcal{H}$ and x another vector of $\mathcal{H}$. By applying formulas 11.62 and 11.63, we get:

$$\begin{aligned}(x|y_1, \cdots, y_{n+1}) &= (x|y_1, \cdots, y_n) + (x|\varepsilon) \\ &= (x|y_1, \cdots, y_n) + \frac{(\varepsilon, x)}{(\varepsilon, \varepsilon)}\, \varepsilon \end{aligned} \tag{11.64}$$

where $\varepsilon = y_{n+1} - (y_{n+1}|y_1, \cdots, y_n)$. Remember that ε is orthogonal to the subspace generated by $(y_1, \cdots, y_n)$.

Formula 11.64 is the basis for recursive formulas.

11.6.1 The Kalman filter

Consider the system described by two sequences of centered random variables X_n and Y_n. In practice, X_n represents a non-observed parameter, also called a *hidden* parameter, and Y_n represents an observation. The system's variables are assumed to obey the two equations 11.65:

$$\begin{cases} X_{n+1} = a_n X_n + B_n \\ Y_n = c_n X_n + U_n \end{cases} \tag{11.65}$$

$\{a_n\}$ and $\{c_n\}$ are two sequences of scalars. $\{B_n\}$ and $\{U_n\}$ are two centered Gaussian sequences, independent of each other, with the variances[3] $\sigma_B^2(n)$ and $\sigma_U^2(n)$ respectively. In particular, $(B_n, Y_k) = \mathbb{E}\{B_n Y_k\} = 0$ and $(U_n, X_k) = \mathbb{E}\{U_n X_k\} = 0$. The first equation of 11.65 describes the evolution of the hidden parameter.

We wish to calculate in a recursive way the orthogonal projection of X_n onto the subspace generated by the n random variables $Y_1, \ldots, Y_n$. We will be using the following notations:

$$\begin{aligned} X_{n|n} &= (X_n | Y_1, \cdots, Y_n) \\ X_{n+1|n} &= (X_{n+1} | Y_1, \cdots, Y_n) \end{aligned}$$

If we use the projection property of the direct sum of two complementary subspaces, we get:

$$\begin{aligned} X_{n+1|n+1} &= (X_{n+1} | Y_1, \cdots, Y_n) + (X_{n+1} | \varepsilon_{n+1}) \\ &= X_{n+1|n} + G_{n+1} \varepsilon_{n+1} \end{aligned} \tag{11.66}$$

where we have assumed:

$$\varepsilon_{n+1} = Y_{n+1} - (Y_{n+1} | Y_1, \cdots, Y_n) \quad \text{and} \quad G_{n+1} = \frac{(X_{n+1}, \varepsilon_{n+1})}{||\varepsilon_{n+1}||^2}$$

If we replace X_{n+1} with $a_n X_n + B_n$, then use the fact that orthogonal projection is linear, as well as the hypothesis stating that B_n is orthogonal to the random variables $Y_1, \ldots, Y_n$, we get:

$$X_{n+1|n} = (X_{n+1} | Y_1, \cdots, Y_n) = a_n (X_n | Y_1, \cdots, Y_n) = a_n X_{n|n} \tag{11.67}$$

If we replace Y_{n+1} with $c_{n+1} X_{n+1} + U_{n+1}$ and use the hypothesis stating that U_{n+1} and B_{n+1} are orthogonal to $Y_1, \ldots, Y_n$, we get:

$$\begin{aligned} (Y_{n+1} | Y_1, \cdots, Y_n) &= c_{n+1} (X_{n+1} | Y_1, \cdots, Y_n) \\ &= c_{n+1} X_{n+1|n} \\ &= c_{n+1} a_{n+1} X_{n|n} + \underbrace{(B_{n+1} | Y_1, \cdots, Y_n)}_{=0} \\ &= c_{n+1} a_{n+1} X_{n|n} \end{aligned}$$

[3] Notice that the variances depend on n.

If we replace this in 11.66, and use the expression of ε_{n+1}, we also get:

$$X_{n+1|n+1} = X_{n+1|n} + G_{n+1}(Y_{n+1} - c_{n+1}a_{n+1}X_{n|n})$$

As a conclusion, using 11.67:

$$X_{n+1|n+1} = a_n X_{n|n} + G_{n+1}(Y_{n+1} - c_{n+1}a_{n+1}X_{n|n}) \tag{11.68}$$

We will now determine the recurrence relation for G_{n+1}. We have:

$$\begin{aligned} \varepsilon_{n+1} &= \underbrace{c_{n+1}X_{n+1} + U_{n+1}}_{=Y_{n+1}} - c_{n+1}X_{n+1|n} \\ &= c_{n+1}(X_{n+1} - X_{n+1|n}) + U_{n+1} \end{aligned}$$

Let $K_n = (X_{n+1} - X_{n+1|n}, X_{n+1} - X_{n+1|n})$. Because $(X_{n+1} - X_{n+1|n})$ and U_{n+1} are orthogonal, we can write:

$$||\varepsilon_{n+1}||^2 = c_{n+1}K_n c_{n+1} + \sigma_U^2(n+1)$$

Furthermore:

$$\begin{aligned} (X_{n+1}, \varepsilon_{n+1}) &= (X_{n+1}, X_{n+1} - X_{n+1|n})c_{n+1} + \underbrace{(X_{n+1}, U_{n+1})}_{=0} \\ &= (X_{n+1} - X_{n+1|n}, X_{n+1} - X_{n+1|n})c_{n+1} \\ &= K_n c_{n+1} \end{aligned}$$

And hence:

$$G_{n+1} = \frac{K_n c_{n+1}}{\sigma_U^2(n+1) + c_{n+1}K_n c_{n+1}} \tag{11.69}$$

Let us now determine the expression of K_n. We have:

$$\begin{aligned} X_{n+1} - X_{n+1|n} &= (a_n X_n + B_n) - a_n X_{n|n} \\ &= a_n X_n + B_n - (a_n X_{n|n-1} + a_n G_n \varepsilon_n) \\ &= a_n(X_n - X_{n|n-1}) + B_n + G_n a_n(c_n(X_n - X_{n|n-1}) + U_n) \\ &= a_n(1 - G_n c_n)(X_n - X_{n|n-1}) + B_n + G_n a_n U_n \end{aligned}$$

This leads us to:

$$\begin{aligned} K_n &= a_n(1 - G_n c_n)K_{n-1}(1 - G_n c_n)a_n \\ &\quad + \sigma_B^2(n) + G_n a_n c_n \sigma_U^2(n) G_n a_n \end{aligned} \tag{11.70}$$

If we group expressions 11.68, 11.69 and 11.70 together, we get the following algorithm called the *Kalman algorithm*:

$$\left\{\begin{array}{lcl} G_n & = & \dfrac{K_{n-1}c_n}{\sigma_U^2(n) + c_n^2 K_{n-1}} \\ X_{n|n} & = & a_{n-1}X_{n-1|n-1} + G_n\left(Y_n - c_n a_n X_{n-1|n-1}\right) \\ K_n & = & a_n^2(1-G_n c_n)^2 K_{n-1} + G_n^2 a_n^2 \sigma_U^2(n) + \sigma_B^2(n) \end{array}\right. \tag{11.71}$$

with the initial conditions $x_{0|0} = \mathbb{E}\{X_0\}$ and $K_0 = \mathbb{E}\left\{X_0^2\right\}$.

11.6.2 The vector case

In the case where the observations and the hidden variables are vectors rather than scalars, the model can be written simply by replacing the scalars with matrices of adequate size:

$$\left\{\begin{array}{l} \mathbf{x}_{n+1} = \mathbf{A}_n\mathbf{x}_n + \mathbf{b}_n \\ \mathbf{y}_n = \mathbf{C}_n\mathbf{x}_n + \mathbf{u}_n \end{array}\right. \tag{11.72}$$

Example 11.5 Consider a vehicle moving along a straight line at the constant speed v. Starting at an initial position d_0, the position at the time n is given by $d(n) = d_0 + vn$. This motion equation can also be written:

$$\left\{\begin{array}{l} d(n+1) = d(n) + v(n) \\ v(n+1) = v(n) \end{array}\right.$$

with the initial conditions $d(0) = d_0$ and $v(0) = v$. Notice that the second equation of the system is reduced to $v(n) = v$ if we assume that the vehicle has a constant speed. But we have little faith in this hypothesis of a constant speed. This is why the possible variability is taken into account by assuming that:

$$\left\{\begin{array}{l} d(n+1) = d(n) + v(n) \\ v(n+1) = v(n) + b_2(n) \end{array}\right.$$

where $b_2(n)$ is a random process acting as a *modeling noise*.

Hence the evolution equation has the expression, in matrix form:

$$\begin{bmatrix} d(n+1) \\ v(n+1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d(n) \\ v(n) \end{bmatrix} + \begin{bmatrix} 0 \\ b_2(n) \end{bmatrix}$$

Let us now assume that the position $d(n)$ is observed at the output of a noised device delivering the value $y(n) = d(n) + u(n)$, where $u(n)$ is a random process used as a model for the *measurement noise.* The set comprising the equation that describes the motion and the one that describes the observation leads to the following system of equations:

$$\begin{cases} \begin{bmatrix} d(n+1) \\ v(n+1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d(n) \\ v(n) \end{bmatrix} + \begin{bmatrix} 0 \\ b_2(n) \end{bmatrix} \\ y(n) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} d(n) \\ v(n) \end{bmatrix} + u(n) \end{cases}$$

If we let $\mathbf{x}_n = [d(n) \quad v(n)]^T$, and if we write the expression of the observation $\mathbf{y}_n$ in vector form, we get as expected the expression of 11.72:

$$\begin{cases} \mathbf{x}_{n+1} = \mathbf{A}_n \mathbf{x}_n + \mathbf{b}_n \\ \mathbf{y}_n = \mathbf{C}_n \mathbf{x}_n + \mathbf{u}_n \end{cases}$$

The $(m \times 1)$ vector $\mathbf{x}_n$ is called the *state vector*, or just the *state*, and the first equation is called the *state equation.* The $(p \times 1)$ vector $\mathbf{y}_n$ is the *measurement vector* and the second equation is called the *observation equation.* The sizes of the matrices $\mathbf{A}_n$ and $\mathbf{C}_n$ are $(m \times m)$ and $(p \times m)$ respectively, and may depend on n.

The respective covariance matrices of $\mathbf{b}_n$ and $\mathbf{u}_n$ are denoted by $\mathbf{R}_b(n) = \mathbb{E}\left\{\mathbf{b}_n \mathbf{b}_n^T\right\}$ and $\mathbf{R}_u(n) = \mathbb{E}\left\{\mathbf{u}_n \mathbf{u}_n^T\right\}$. A calculation in every way similar to the one we did previously in the scalar case leads to the following algorithm:

Initial values:
$\mathbf{x}_{0|0} = \mathbb{E}\{\mathbf{x}_0\}$ *and* $\mathbf{K}_0 = \mathbb{E}\left\{\mathbf{x}_0 \mathbf{x}_0^T\right\}$
Repeat:

$$\begin{aligned} \mathbf{G}_n &= \mathbf{K}_{n-1} \mathbf{C}_n^T \left(\mathbf{C}_n \mathbf{K}_{n-1} \mathbf{C}_n^T + \mathbf{R}_u(n)\right)^{-1} \\ \mathbf{x}_{n|n} &= \mathbf{A}_{n-1} \mathbf{x}_{n-1|n-1} + \mathbf{G}_n \left(\mathbf{y}_n - \mathbf{C}_n \mathbf{A}_n \mathbf{x}_{n-1|n-1}\right) \\ \mathbf{K}_n &= \mathbf{A}_n \left(\mathbf{I} - \mathbf{G}_n \mathbf{C}_n\right) \mathbf{K}_{n-1} \left(\mathbf{I} - \mathbf{G}_n \mathbf{C}_n\right)^T \mathbf{A}_n^T \cdots \\ &\qquad\qquad + \mathbf{A}_n \mathbf{G}_n \mathbf{R}_u(n) \mathbf{G}_n^T \mathbf{A}_n^T + \mathbf{R}_b(n) \end{aligned}$$

We will come back to the Kalman filter in paragraph 12.14.

Chapter 12

Selected Topics

12.1 Simulation of continuous-time systems

12.1.1 Simulation by approximation

Design methods based on continuous-discrete time changes actually consist of constructing a discrete-time simulator of a linear differential equation. This method provides satisfying results because the simulated systems are linear. Many precautions would have been needed had they not.

Exercise 12.1 illustrates the implementation of an RC filter simulator subjected to a periodic input.

Exercise 12.1 (Full-wave rectifier and simulation)
Consider a full-wave rectifier followed by an RC filter (Figure 12.1).

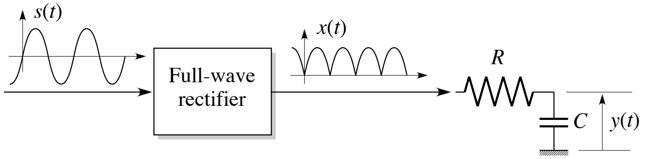


Figure 12.1 – *Full-wave rectifier*

1. The input signal $s(t) = A\sin(2\pi F_0 t)$ with $F_0 = 50$ Hz is fed to the rectifier. Determine the Fourier series expansion of the rectified signal $x(t) = |s(t)|$. What is the amplitude of the continuous component of $x(t)$?

2. The output voltage $y(t)$ of the RC filter verifies the differential equation:

$$RC\frac{dy(t)}{dt} + y(t) = x(t)$$

Using the properties of the Fourier transform, determine the expression of this filter's complex gain $H(F)$.

3. $1/RC$ is chosen to be much greater than F_0 so that only the continuous component and the first harmonics remain in the output signal. What is, in this case, the expression of $y(t)$?

4. We wish to simulate this system. In order to do so, we perform a sharp enough discretization of time by choosing a sampling frequency $F_s = 1/T_s =$ 5,000 Hz much greater than the input sine's frequency.

   We define $x_s(n) = x(nT_s)$ $y_s(n) = y(nT_s)$. By using the Euler approximation $dy/dt \simeq F_s(y(nT_s) - y((n-1)T_s))$, show that the differential equation is equivalent to the recursive equation $y_s(n) + (\tau - 1)y_s(n-1) = \tau x_s(n)$. Determine the expression of τ as a function of RC and T_s.

5. Write a program that simulates the system's output voltage when the input is a sinusoidal voltage with a frequency of 50 Hz and an RMS (Root-Mean-Square) voltage of 220V. The filter's time constant is $RC = 0.01$ s. The `filter` function will be used to generate the output signal.

12.1.2 Exact model simulation

Consider a continuous-time filter with a frequency response that tends to 0 when the frequency tends to $+\infty$ (this is called a *strictly proper filter*). The relation between the input $i(t)$ and the output $o(t)$ is then assumed to be described by a constant coefficient linear differential equation. This system can be represented with the use of *state equations* as follows:

$$\begin{cases} \dfrac{d\mathbf{x}(t)}{dt} = \mathbf{A}\mathbf{x}(t) + \mathbf{b}i(t) \\ o(t) = \mathbf{c}^T\mathbf{x}(t) + di(t) \end{cases} \tag{12.1}$$

where $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{x}(t)$ are $n \times 1$ vectors and d is a scalar, equal to zero. $\mathbf{A} = [a_{ij}]$ is an $n \times n$ matrix with its $[a_{ij}]$ time-independent. $\mathbf{x}(t)$ is called the *state vector* of this representation, which is far from being the only possible one.

It can be shown that the solution to the first of the system's equations is:

$$\mathbf{x}(t) = e^{\mathbf{A}(t-t_0)}\mathbf{x}(t_0) + \int_{t_0}^{t} e^{\mathbf{A}(t-u)}\mathbf{b}i(u)du \tag{12.2}$$

where the matrix exponential (the MATLAB® function `expm`) is given by:

$$e^{\mathbf{A}t} = \mathbf{I} + \frac{t}{1!}\mathbf{A} + \frac{t^2}{2!}\mathbf{A}^2 + \cdots + \frac{t^n}{n!}\mathbf{A}^n + \cdots$$

Given this definition, you can check that:

$$\frac{de^{\mathbf{A}t}}{dt} = \mathbf{A}e^{\mathbf{A}t} = e^{\mathbf{A}t}\mathbf{A}$$

and:

$$\mathbf{A}\int_0^t e^{\mathbf{A}u}du = \int_0^t e^{\mathbf{A}u}du \times \mathbf{A} = \left(e^{\mathbf{A}t} - \mathbf{I}\right)$$

The input-output relation is obtained by applying the Laplace transform to the system 12.1. We get:

$$O(s) = \mathbf{c}^T(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{b}I(s) \tag{12.3}$$

where $I(s)$ and $O(s)$ are the Laplace transforms of $i(t)$ and $o(t)$ respectively.

Example 12.1 (Second order systems) Consider a continuous-time system, with the input $i(t)$ and the output $o(t)$, both scalar, described by the differential equation:

$$\frac{d^2o(t)}{dt^2} + a_1\frac{do(t)}{dt} + a_2o(t) = i(t) \tag{12.4}$$

Let:

$$\mathbf{x}(t) = \begin{bmatrix} o(t) & \dfrac{do(t)}{dt} \end{bmatrix}^T$$

With this choice of the state vector, equation 12.4 leads us to a state representation:

$$\begin{cases} \dfrac{d\mathbf{x}(t)}{dt} = \begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} i(t) \\ o(t) = [1 \quad 0]\mathbf{x}(t) \end{cases}$$

similar to that of 12.1. We have to check that the input-output relation is:

$$O(s) = \frac{1}{s^2 + a_1s + a_2}I(s)$$

by applying the Laplace transform to 12.4 (the initial conditions are assumed to be equal to zero) or by 12.3.

Exercise 12.2 illustrates the implementation, in the particular case of a simulation based on a state representation. Such a simulation is particularly useful in automatic control where the filter's output (called a compensator in this context) is applied to a continuous-time process through a digital-to-analog converter.

Exercise 12.2 (Simulation in the presence of a ZOH)

We wish to simulate the behavior of a continuous-time system described by a state representation. The input signal is obtained with a ZOH DAC that maintains the input value during the sampling period (Figure 12.2).

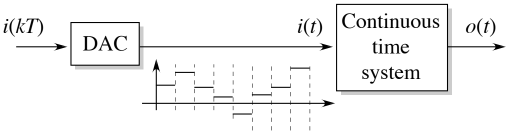


Figure 12.2 – *Continuous-time system fed through a ZOH DAC*

1. Give, based on expression 12.2, the relation between the state vector at the time $(k+1)T$ and the state vector at the time kT. Show that we can find a relation similar to:

   $$\mathbf{x}((k+1)T) = \mathbf{\Phi}(T)\mathbf{x}(kT) + i(kT)\mathbf{\Psi}(T)\mathbf{b}$$

   where $\mathbf{\Psi}(T)$ is obtained as a k-independent integral of $\mathbf{\Phi}(t) = e^{\mathbf{A}t}$. Notice that the integral $\int_0^t e^{\mathbf{A}u}du$ does not require the calculation of $\mathbf{A}^{-1}$, the invertibility of which is not certain. It can actually be calculated directly by taking the exponential $\exp(\mathbf{A}_e t)$ with:

   $$\mathbf{A}_e = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \underline{0}^T & 0 \end{bmatrix}$$

2. Give $\mathbf{x}(t)$ as a function of $\mathbf{\Phi}(t)$ and $\mathbf{\Psi}(t)$, for $t \in (kT, (k+1)T)$.

3. Consider the filter defined by the differential equation 12.5:

   $$\frac{d^2o(t)}{dt^2} + 1.4\frac{do(t)}{dt} + o(t) = \mathbb{1}(t \in [0, +\infty)) \tag{12.5}$$

   with the unit step as its input. Starting with the initial conditions:

   $$o(0) = 0 \quad \left.\frac{do(t)}{dt}\right|_{t=0} = 0$$

   simulate the response to the unit step.

4. For the same system, apply the bilinear transform to perform the continuous-discrete change, and find the same response to the unit step.

Notice that, in this exercise, the unit step response given in question 3 is correct. This means that the calculated values of the output sequence coincide with the values $o(nT)$ for all T. However, the answer found with the bilinear transform is only an approximation.

Exercise 12.3 (Non-minimal system)
The use of linear, time-invariant, continuous-time models in the form of state representations has led us the following expressions:

$$\left\{\begin{array}{l} \mathbf{A} = \begin{bmatrix} -11/4 & -11/8 & -5/4 \\ 27/4 & 11/8 & 21/4 \\ 15/8 & 19/16 & 5/8 \end{bmatrix} \\ \mathbf{b} = \begin{bmatrix} 1 \\ -1 \\ -1/2 \end{bmatrix}, \mathbf{c}^T = \begin{bmatrix} 3/8 & 1/2 & -1/4 \end{bmatrix}, d = 0 \end{array}\right. \tag{12.6}$$

1. We wish to "digitally" simulate the response of this system when fed by a unit step through a ZOH. Perform this simulation using the results of the previous exercise, for several sampling values and for a minimum simulation duration of 10 s.

2. Observe the evolution of the $||.||_\infty$ norm of the state vector during the simulation. What conjecture can be made concerning the stability?

3. Find the transfer function (Laplace transform) associated with the system 12.6. What can be said of the stability? You may find the comments in paragraph 4.4.3 on the role played by initial conditions in a system's behavior useful for answering this question (we have the same result in continuous and discrete time cases).

4. Perform the simulation for a period of about 3 minutes. What happens? What is this phenomenon caused by? It can be verified that the discrete TF can be calculated by `poly(phi-psib*c')/poly(phi) - 1`.

12.2 Dual Tone Multi-Frequency (DTMF)

On a Dual Tone Multi-Frequency (DTMF) phone keyboard, each key is associated with the sending of a signal. This signal is the sum of two sines, the frequencies (in Hz) of which are given in the correspondence Table 12.1.

This means that when you dial "5" on your phone, the signal $x(t) = \cos(2\pi \times 1{,}336 \times t) + \cos(2\pi \times 770 \times t)$ is sent through the phone line.

Hz	1,209	1,336	1,477
697	1	2	3
770	4	5	6
852	7	8	9
941	$\star$	0	#

Table 12.1 – Frequency correspondence table

COMMENTS:

- These frequencies belong to the (300 Hz - 3,400 Hz) band, the phone band for the switched network (fixed phones). The frequencies associated to the columns are all greater than those associated with the lines. This layout can help to find the phone number using the signal. Finally, the frequencies were chosen so as not to have frequency ratios equal to integers. As we have seen, a non-linear operation can cause multiples (harmonics) of the fundamental frequency to appear, causing some confusion.

- The keyboard is designed to always send signals for periods longer than $\tau_1 = 65$ milliseconds. This value was chosen so that the two frequencies contained in the signal could easily be separated. In the worst case, the difference in frequency is $\Delta F_{\text{min}} = 1{,}209 - 941 = 268$ Hz (corresponding to the $\star$ key), therefore τ_1 needs to be such that $\Delta F_{\text{min}}\tau_1 \gg 1$. With the values that were chosen, $\Delta F_{\text{min}}\tau_1 > 17$.

- Finally, it must be possible to tell the difference between the number C being sent for a duration of T and the number CC being sent for the same duration. This is why, after each key is released, a zero signal is sent for at least 80 ms (even if you can dial faster than that!).

Example 12.2 (DTMF signal processing)
We are going to try to find a 10 digit phone number using the signal sent by the phone. We will start by sampling the signal at a frequency of 8,000 samples per second, a speed much higher than twice the highest frequency, that is $2 \times 1{,}477 = 2{,}954$ Hz.

The following program creates such a signal:

```
%===== GENEKEY.M
clear
Fs=8000;              % Sampling freq.
tel='0145817178'; lt=length(tel); % Seq. of numbers
%===== Coding table
```

```
keys='123456789*0#'; nbkeys=length(keys);
FreqB=[697 770 852 941]; FreqH=[1209 1336 1477];
Freqskeys=...
   [FreqB(1) FreqH(1); FreqB(1) FreqH(2); % 1 et 2
    FreqB(1) FreqH(3); FreqB(2) FreqH(1); % 3 et 4
    FreqB(2) FreqH(2); FreqB(2) FreqH(3); % 5 et 6
    FreqB(3) FreqH(1); FreqB(3) FreqH(2); % 7 et 8
    FreqB(3) FreqH(3); FreqB(4) FreqH(1); % 9 et *
    FreqB(4) FreqH(2); FreqB(4) FreqH(3)];% 0 et #
%===== Constraints
tton=0.065; tsil=0.080;            % in seconds
%==== Construction of the seq. of frequencies
Freqs=zeros(lt,2);
for ii=1:lt
    ind=find(keys==tel(ii));       % Test of the number
    Freqs(ii,:)=Freqskeys(ind,:);  % Associated Freq.
end
freqs=Freqs/Fs;                    % Normalized freq.
%===== Construction of the signal
y=zeros(100+fix(100*rand),1);      % Starting with signal=0
dton=fix(1000*rand(lt,1)+tton*Fs); % Number duration
dsil=fix(1000*rand(lt,1)+tsil*Fs); % Silence duration
for ii=1:lt
    sigu=cos(2*pi*(0:dton(ii))'*freqs(ii,:))*ones(2,1);
    y=[y;sigu;zeros(dsil(ii),1)];
end
%===== Some noise is added
lx=length(y); py=y'*y/lx;
SNR=30; pb=py*10^(-SNR/10); x=y+sqrt(pb)*randn(lx,1);
%===== Plotting of the signal
tps=(0:lx-1)/Fs; plot(tps,x); grid
set(gca,'xlim',[0 (lx-1)/Fs])
```

In order to simulate the perturbations on an actual phone call, the program adds noise created by `sqrt(pb)*randn(L,1)`. `SNR` is the signal-to-noise ratio (in dB) chosen equal to 30 dB. The resulting signal is shown in Figure 12.3.

We are going to find the 10 digit number in this signal in two steps. First, we will determine the beginning and the end of the signal's active zones, then we will analyze each of the intervals to extract the frequencies and therefore the corresponding digit. To determine the beginning and the end of the active zones of the signal, we are going to estimate the "instantaneous power" and compare it to a threshold value. We will see later on as we study random phenomena what we mean exactly by "estimating the instantaneous power". Here, we will merely be considering the quantity:

$$P_n = \frac{1}{N} \sum_{k=n-N+1}^{n} x_k^2 \tag{12.7}$$

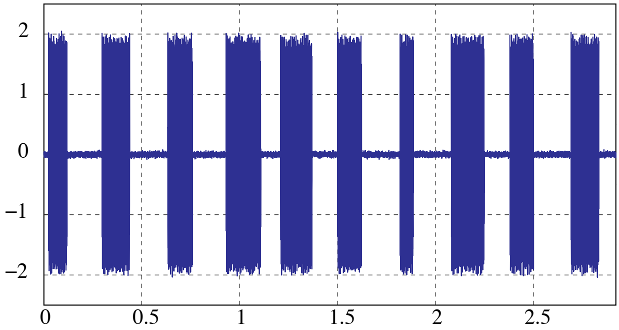


Figure 12.3 – *DTMF Signal*

which gives a relevant indication on the signal's fluctuations. The choice of N is done as a compromise. Consider, for example,the signal $x(n)$ represented in Figure 12.3. If N is very small, P_n will be very close to the amplitude x_n^2. The risk would be to make the conclusion that the power is equal to zero whenever the amplitude is close to 0 (which happens every period). If, on the contrary, N is very high, we might include a silence and miss the beginning or the end of an active part. Quantitatively, N must therefore be much greater than the longest of the periods of the active parts, and much smaller than the duration of the wanted signal, that is to say 65 ms. This can be expressed:

$$\frac{F_s}{697} \ll N \ll 65 \times 10^{-3} F_s$$

For $F_s = 8{,}000$, and with $N = 100$, this double inequality is satisfied.

1. Write a program that measures the "instantaneous power" and determines the beginning and the end of the signals associated with a digit.

2. Write a program that determines the digit associated with each portion of the signal.

HINT:

1. The expression 12.7 for the power can be seen as the result of the filtering of the positive signal $y_n = x_n^2$ by the filter with the finite impulse response $h(k) = 1/N$ for $k \in \{0, \dots, N-1\}$, and 0 otherwise. The `filter` function, a function we will study in detail in Chapter 4, is used as follows to achieve this filtering operation:

   ```
   hpb=ones(N,1)/N; pn=filter(hpb,1,x .* x);
   ```

   The program that performs the complete signal activity detection operation is the following:

```
%===== DETECTKEY.M
% input x=DTMF signal
N=100; hpb=ones(N,1)/N;    % Estimation of the instantaneous
pn=filter(hpb,1,(x .* x)); % power for N points
pmax=max(pn); mthresh=0.5*pmax;
marker=(pn>mthresh); lmarker=length(marker);
begend=marker(2:lmarker)-marker(1:lmarker-1); % diff(marker)
begs=find(begend==+1)-N/2; % Start indices
ends=find(begend==-1)-N/2; % Stop indices
%===== Plotting the signal
subplot(211),plot(tps,x); set(gca,'xlim',[0 (lx-1)/Fs])
%===== Plotting the transitions
hold on
for ii=1:length(begs)
    plot(begs(ii)/Fs*[1 1],4*[-1 1],'-r');
end
for ii=1:length(ends)
    plot(ends(ii)/Fs*[1 1],4*[-1 1],'-');
end
hold off; grid
%===== Plotting of the instant. power and threshold
subplot(212); plot(tps,pn)
set(gca,'xlim',[0 (lx-1)/Fs],'ylim',[0 1.2*pmax])
hold on; plot([0 (lx-1)/Fs],[mthresh mthresh],'-');
hold off; grid
```

The filter output is compared to a threshold chosen equal to half of the maximum instantaneous power. Using the logical expression **pn>mthresh**, we then determine the sequence of the parts of the signal where P_n is above the threshold value. By subtracting this sequence to itself translated by 1, we get of sequence of values, where $+1$ indicates the beginning of a signal and -1 the end of a signal. This operation corresponds to a numerical derivative that could just as well have been written **debfin=diff(marque)**. In expression 12.7, the calculated power corresponds to the signal portion going from the indices $(n-N+1)$ to n. It is therefore better to consider this measure as the median position $n-N/2$. This is why we subtracted $N/2$ to the positions that were found, which is done by the two following program lines:

```
begs=find(begend==+1)-N/2;
ends=find(begend==-1)-N/2;
```

We represented in Figure 12.4 the estimated instantaneous power.

2. To determine the digit based on an active portion of the signal, the two frequencies have to be extracted. The correspondence table provides the corresponding number. All the frequencies extracted are already known,

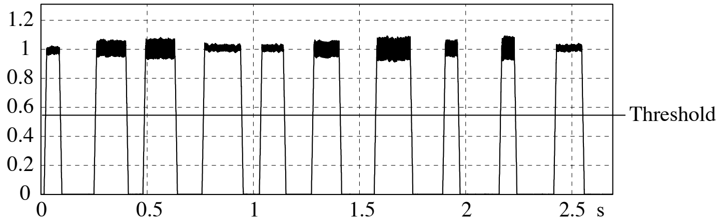


Figure 12.4 – *Estimated instantaneous power*

so there is no use doing a spectral study on the entire frequency axis. It is sufficient to evaluate the spectrum at frequencies that might be contained in the signal, by calculating these four quantities:

$$Q = \left|\sum_n x(n)e^{2j\pi f n}\right| \tag{12.8}$$

for $f \in \{697/F_s, 770/F_s, 852/F_s, 941/F_s\}$, with F_s = 8,000 Hz, and to choose the frequency that corresponds to the highest value. The same is done with the group of 3 high frequencies.

The program **detectnum.m** determines the phone number associated with the signal: ■

```
%===== DETECTNUM.M
nbDigitdet=min([length(begs) length(ends)]);
foundNum=[];
for tt=1:nbDigitdet
    QB=zeros(4,1); QH=zeros(3,1);
    sig=x(begs(tt):ends(tt)); % Signal associated with a number
    lsig=length(sig);
    %===== For each of the 4 freq., calculating the correlation
    for ii=1:length(FreqB)    % For each freq.
        ps=sig .* exp(2*j*pi*FreqB(ii)*(1:lsig)'/Fs);
        QB(ii)=abs(sum(ps));
    end
    %===== For each of the 3 freq., calculating the correlation
    for ii=1:length(FreqH)
        ps=sig .* exp(2*j*pi*FreqH(ii)*(1:lsig)'/Fs);
        QH(ii)=abs(sum(ps));
    end
    %===== Maxima
    [bid, indB]=max(QB); [bid, indH]=max(QH);
    detF=[FreqB(indB) FreqH(indH)];
    %===== Table Look-up
    jj=1;
    while sum(Freqskeys(jj,:)~=detF), jj=jj+1; end
    foundNum=[foundNum keys(jj)];
end
disp(sprintf('******* The number is : %s',foundNum))
```

12.3 Speech processing

Speech is an important field for the applications of digital signal processing. This first paragraph deals with how transporting and storing a signal can be facilitated by analyzing and compressing it.

12.3.1 A speech signal model

Overview

The first issue is the choice of the sampling frequency. When it comes to telephone communications, there usually are two constraints: the message must be comprehensible, and must make it possible to identify the person speaking. These constraints mean that the frequency band can be restrained to the [300-3,400] Hz interval, which implies a Nyquist frequency of 8 kHz. Figure 12.5 shows 2,000 values, or 0.25 s of a speech signal sampled at that frequency.

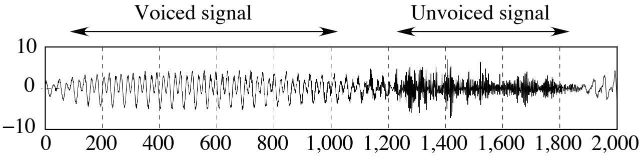


Figure 12.5 – *Speech signal sampled at 8,000 Hz. The* x*-coordinates correspond to the number of samples*

This 300-3,400 Hz band is called the *telephone-band.* Of course, the larger the band, the better the quality. However, as the width of the band increases, so does the amount of informations sent per unit of time!

In order to enhance the quality, a larger band is considered that goes from 150 Hz up to 7 kHz; this band is devoted to the *wideband* digital handsets telephones. It works well for speech signals but is still insufficient for musical signals. In the case of music, usually two qualities are considered: the *FM band* (short for Frequency Modulation) quality used for frequency modulated radio broadcasts, which goes up to 15 kHz, and the *HIFI band* quality, used for example for compact discs, which goes up to 22 kHz. In any case, the speech signal must be sampled, according to the sampling theorem, at a frequency at least twice that of the band used.

Quality	Maximum frequency	Sampling frequency
Telephone-band	300-3,400 Hz	8,000 samples/s
Wideband	150-7,000 Hz	16,000 samples/s
FM band	50 Hz-15,000 Hz	32,000 samples/s
HIFI band	<22,050 Hz	44,110 samples/s

A typology of vocal sounds

Consider the part of the signal shown in Figure 12.5. As you can see, there are two clearly distinct sections corresponding to two types of sounds:

– The sounds that have the aspect of a harmonic vibration and that are said to be *voiced*. Vowels are a perfect illustration of this type of sound. An example is shown in Figure 12.5 in the window with the indices from 0 to 1,200.

– The sounds we interpret more as noise, and that are said to be *unvoiced*. An example is shown in Figure 12.5 in the window with the indices from 1,200 to 1,800.

Vowels generally last longer than consonants. They can easily be recognized by their harmonic aspect. Consonants are divided in the following categories:

– the *nasal* consonants/m/, /n/...for which the “oral cavity + pharynx” system forms a closed resonant cavity, with the air going the nostrils;

– the *unvoiced fricatives* /f/, /s/, /ch/... produced by turbulence of a continuous air flow in the oral cavity. The cavity is divided in two subcavities, the one in the back causing the “zeros” in the transfer function;

– the *voiced fricatives* /v/, /z/... which can be described in the same way as the unvoiced fricatives but with vibrations of the vocal folds;

– the *voice plosives* b/, /d/...which are transitions caused by the sudden opening of the oral cavity following a rise in pressure. They strongly depend on the vowels they are pronounced with;

– the *voiced plosives* /p/, /t/, etc.

Figure 12.6 illustrates the case of the sounds “sh” and “ee”.

The AR model of speech production

The production of sounds is a very complex phenomenon that cannot be easily described by a model. It is bound to the anatomy of the vocal tract, represented in Figure 12.7.

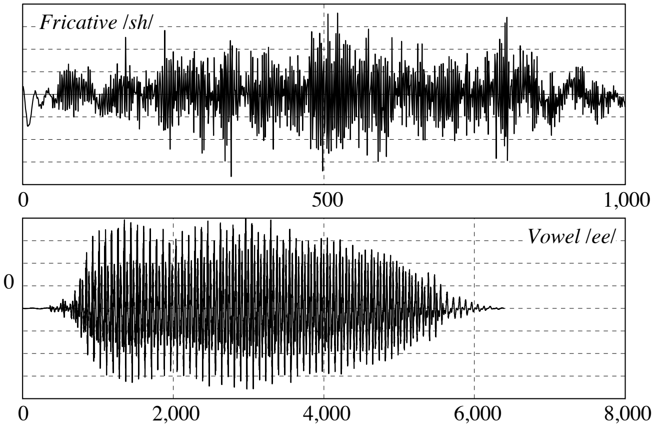


Figure 12.6 – *Temporal shapes of a speech signal sampled at 8,000 Hz: top graph: an unvoiced sound; bottom graph: voiced sound*

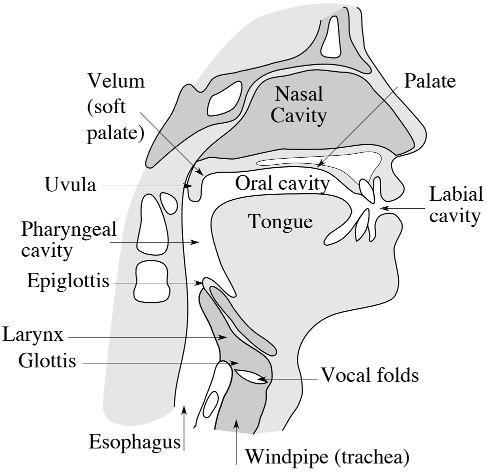


Figure 12.7 – *Anatomy of the vocal tract*

A functional represents [35] is shown in Figure 12.8. The vocal tract is simplified as a series of cavities, the shapes of which change with time as air, coming from the lungs, flows through them.

Studies conducted on the vocal tract show that the two type of sounds, voiced and unvoiced, can be described using as a model the output of a all pole linear filter of the type $1/A(z)$, the order of which is between 10 and 50, and with, as the input:

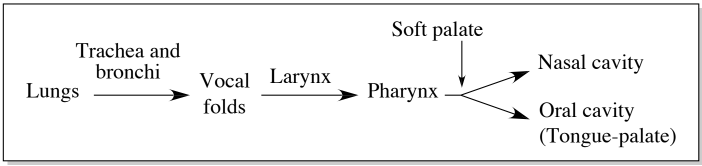


Figure 12.8 – *Elements of the vocal tract*

– a white noise for unvoiced sounds;

– and an impulse sequence for voiced sounds.

The impulse sequence associated with voiced sounds corresponds the derivative of the volume of the air flowing through the glottis (the space between the vocal cords) which opens and closes periodically, with opening phases that last longer than the closing phases. This leads to a sudden decrease, during the closing phase, of the air flow which causes by derivation a very brief negative impulse (see Figure 12.9). Although it is sometimes sufficient to crudely approximate the sequence produced by the glottis by a simple sequence of ideal impulses, there are more sophisticated glottal excitation models [47], such as those of Rosenberg or Liljencrants-Fant (see Figure 12.9).

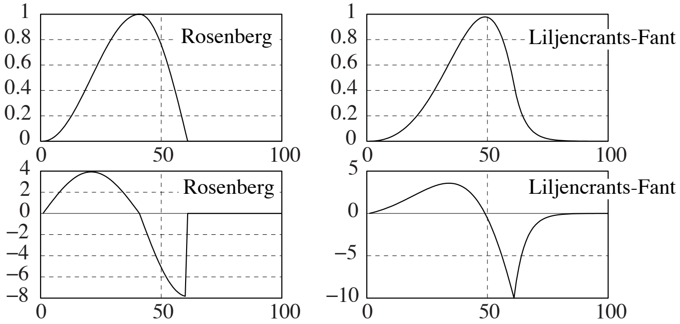


Figure 12.9 – *Typical shapes of the glottal excitation signal. Above figure: air flow model. Bottom figure: derivative of the air flow model*

The fundamental frequency of the periodic sequence of glottal impulses is called the *pitch*. This frequency goes from about 70 Hz for a very low voice to 450 Hz for very high one. For a man, it goes basically from 70 to 200 Hz, for a woman from 140 to 350 Hz, and for a child from 180 to 450 Hz. In any case, for a given person, the pitch varies in the course of a conversation. For voiced sounds, the sequence of periodic impulses produced by the vocal folds acts as a frequency analyzer and cause resonant frequencies to appear in the vocal tract. These frequencies are called *formants.* As you can see in Figure 12.10, 4 formants are found by following the spectrum's envelope shown on the right.

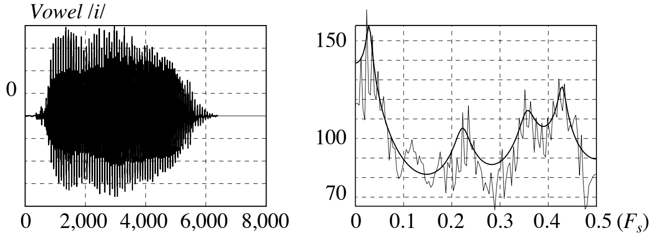


Figure 12.10 – *Temporal shape and spectrum of a speech signal: the graph on the right shows four formants (sampling frequency F_s = 8,000 Hz)*

Let $1/A(z)$ be the transfer function of the filter used as a model for the vocal tract. In the case of an *unvoiced sound*, the input can be seen as a white noise, and therefore the speech signal is an autoregressive process. We can then estimate the coefficients of $A(z)$ from an unvoiced sound window, using the results obtained for AR processes in paragraph 9.2.1 on page 329. Then, when the unvoiced signal is applied to the FIR filter with the transfer function $A(z)$, we get an estimate of the white noise input.

In the case of a *voiced sound*, the glottis signal is assumed to be a sequence of periodic impulses with the pitch frequency F_p. If the vocal tract is described as an all pole filter $1/A(z)$ and if we assume that:

$$g(n) = \sum_k i(n-kM) \approx \sum_k A\delta(n-kM) \tag{12.9}$$

provides a good approximation of the glottis signal with $MT_s \approx 1/F_p$, the voiced signal $s(n)$ is a periodic signal containing the *same* frequencies. If $A(z)$ is a P degree polynomial, $s(n)$ obeys the filtering equation:

$$s(n) + a_1 s(n-1) + \cdots + a_P s(n-P) = g(n) \tag{12.10}$$

Based on N observations, we get:

$$\begin{bmatrix} s(P+1) & s(P) & \cdots & s(1) \\ \vdots & \vdots & \ddots & \vdots \\ s(N) & s(N-1) & \cdots & s(N-P) \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ \vdots \\ a_P \end{bmatrix} = \begin{bmatrix} g(P+1) \\ \vdots \\ \vdots \\ g(N) \end{bmatrix}$$

which is written in matrix form:

$$\begin{bmatrix} \mathrm{s} & \mathbf{S} \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{a} \end{bmatrix} = \mathrm{g} \tag{12.11}$$

$\mathbf{S}$ is a Toeplitz matrix. The vector $\mathbf{g}$ is a periodic vector containing an A followed by $(M-1)$ zeros (see approximation 12.9). Typically, for a pitch frequency of $F_p = 190$ Hz, the pitch period is $T_p \approx 5$ ms. If T_p is assumed to be much greater than the glottal impulse duration (see example 10.3), the vector $\mathbf{g}$ contains almost nothing but zeros. Equation 12.11 can then be written:

$$\begin{bmatrix} \mathrm{s} & \mathbf{S} \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{a} \end{bmatrix} \approx \underline{0}$$

and $\mathbf{a}$ can be estimated using a least squares approach by minimizing this vector. The advantage of this method is its simplicity[1] because it requires neither the estimate of the period M nor the estimate of the phase corresponding to the exact moment when the glottis close. The minimization leads to:

$$\hat{\mathbf{a}} = -(\mathbf{S}^T\mathbf{S})^{-1}\mathbf{S}^T\mathrm{s}$$

an expression similar to the expression $\mathbf{a} = -\mathbf{R}^{-1}\mathbf{r}$ found with the Yule-Walker equation 8.58 which relates the parameters of an AR process with the covariance coefficients. Once $\hat{\mathbf{a}}$ has been estimated, we can then calculate $g(n)$ with the use of expression 12.10, that is simply by feeding the speech signal $\{s(n)\}$ into the input of the FIR filter with the transfer function $A(z)$.

As a conclusion, whether the sound is voiced or not, the estimate of the residual signal, glottis impulses in the first case and white noise in the second, is obtained by filtering the signal by the filter with the transfer function $A(z)$, the coefficients of which are estimated as the parameters of an autoregressive process. The following example allows you to experimentally check the shape of the residual signal for a speech signal.

Example 12.3 (Observation of a speech signal's residual) First record a voiced sound `/ee/` and an unvoiced sound `/sh/` at 8 kHz. Then create a program:

- that estimates the coefficients $\{a_k\}$ of a 20th order all pole model for an analysis window with a duration of 30 ms, or 240 samples;
- that performs the filtering with the transfer function $A(z)$ of the same speech signal block.

Apply this process to both the voiced and the unvoiced sound.

HINT: type the following program:

[1] When the glottal impulses have to be reconstructed with a higher accuracy, for example when it is used to perform a medical diagnosis, the least squares method used here can give insufficient results [35].

```
%===== RESIDUAR.M
clear
load voye ; % Voyel
%load conch; % or consonant
Fs=8000; tbloc=.05;          % Block size (ms)
modorder=20; nb=tbloc*Fs; mtime=(0:nb-1)/Fs;
sigi=ltre(1+1600:nb+1600); sigch=ltrch(1:nb);
for k=1:modorder+1,
   ri(k)=sigi(k:nb)'*sigi(1:nb-k+1)/nb;
   rch(k)=sigch(k:nb)'*sigch(1:nb-k+1)/nb;
end
RRi=toeplitz(ri); asi=-RRi \ [1;zeros(modorder,1)];
RRch=toeplitz(rch); asch=-RRch \ [1;zeros(modorder,1)];
ai=asi/asi(1); resi=filter(ai,1,sigi);
ach=asch/asch(1); resch=filter(ach,1,sigch);
subplot(411); plot(mtime,sigi/max(abs(sigi))); grid
subplot(412); plot(mtime,-resi/max(abs(resi))); grid
subplot(413); plot(mtime,sigch/max(abs(sigch))); grid
subplot(414); plot(mtime,resch/max(abs(resch))); grid
```

The results are shown in Figure 12.11 and 12.12. The graphs give the shape of the residual signal for a voiced sound and for an unvoiced sound. Very short impulses are clearly visible for the residual signal of a voiced signal, with a period of about 10 ms, that is about 100 Hz, corresponding to the closing frequency of the glottis.

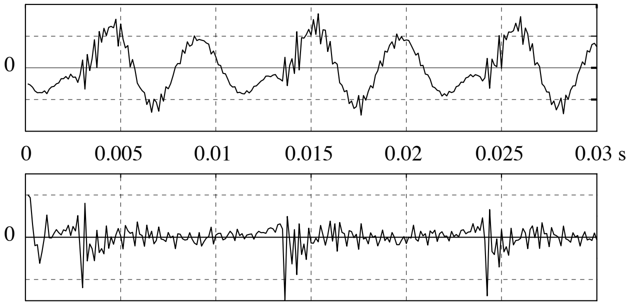


Figure 12.11 – *Temporal shape of a voiced sound (/ee/)and of the residual*

Remember that the *Toeplitz* nature of the estimate of a covariance matrix ensures that the filter with the transfer function $1/A(z)$ is causal and stable (see page 309 on the stability of AR models). However, in the *analysis problem*, which consists of constructing, as we have just done, the input signal based on the observed signal, this property is not crucial since the filter $A(z)$ is a FIR filter, and is therefore stable. On the other hand, in the *synthesis problem*, which consists of reconstructing the speech signal based on the input signal, this property is of course essential. ■

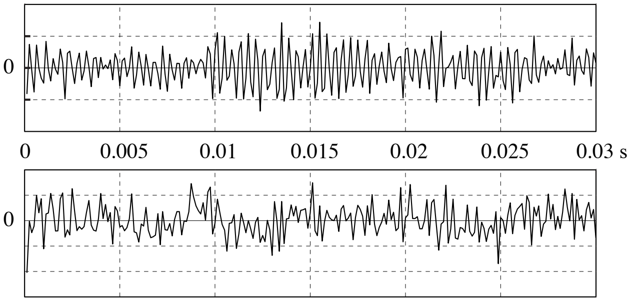


Figure 12.12 – *Temporal shape of an unvoiced sound (/sh/) and of the residual*

12.3.2 Compressing a speech signal

We are going to present in this paragraph an example of speech signal compression. The word compression is used for the digitization of a signal as a bitstream with a rate as low as possible for a given level of distortion. Compression is too difficult a problem to be discussed here thoroughly. The solution we present is merely an introductory example, inspired from a encoder chosen a very wide choice of existing encoders [66, 35].

The simplest form of signal digitization is the b-bit uniform quantization of the values sampled at the frequency F_s. Typically, for $F_s = 8$ kHz and $b = 8$ bits, the result is a 64 kbits/s stream called PCM (*Pulse Code Modulation*).

To achieve compression, in other words to use less bits, while maintaining a low level of distortion, a first idea would be to quantize the difference $\delta(n) = s(n) - s(n-1)$ in place of the samples $s(n)$. The resulting should be smaller than $s(n)$ and hence the same accuracy would be achieved with less bits. This approach, which uses the correlations contained in the consecutive samples can easily be generalized by quantizing the residual signal $\epsilon(n) = s(n) - (\alpha_1 s(n-1) + \cdots + \alpha_P s(n-P))$, where the sequence α_p is the one that minimizes $\mathbb{E}\left\{|\epsilon(n)|^2\right\}$. As you may have noticed, we are once again faced with the linear prediction problem. This approach, called DPCM, for differential PCM has the advantage of being applicable to any signal for what we call *waveform coding.*

A second approach consists of considering the speech creating system and describing it using a model comprising a small number of parameters we are going to estimate. Then, all we have to do is reconstruct the signal from these parameters to obtain a signal that "sounds" like the original one. This type of encoder, called a *vocoder*, does not follow the shape of the original signal. This is why it cannot be used to compress a signal originating for example from a modem operating in the phone frequency band. However, the bitstream is usually lower, for the same quality, than the ones obtained in the first approach. The example we chose belongs to the second category.

The compression principle we decided on is based on representing the speech production system as an all pole filter, the input of which is either a sequence of periodic impulses, in the case of voiced sounds, or a white noise in the case of unvoiced sounds. The result is the synthesis diagram shown in Figure 12.13. With this approach, we assume that the signal is stationary, which is almost true for speech signals with a duration of about 10 to 20 milliseconds.

To have a better idea of this model, consider a signal sampled at 8,000 Hz with 8 bits, which corresponds to a rate of 64,000 bits/s, and assume that for each block of $N = 180$ samples, corresponding to a duration of 22.5 ms, $P = 22$ parameters are extracted. If we use 8 bits to represent each of the 22 parameters, $180 \times 8 = 1{,}440$ bits are replaced with $22 \times 8 = 176$ bits, which means we have a compression factor of about 8. In terms of rate, everything happens as if only 1 bit of each sample was kept. This would be the rate obtained if, for example, we only kept the sign bit of the samples $s(n)$. To compare this with the result obtained in exercise 12.4, you can run the following program:

```
signs=sign(s);
soundsc(signs,8000);
```

where `s` represents the sample sequence of a speech signal sampled at 8,000 Hz.

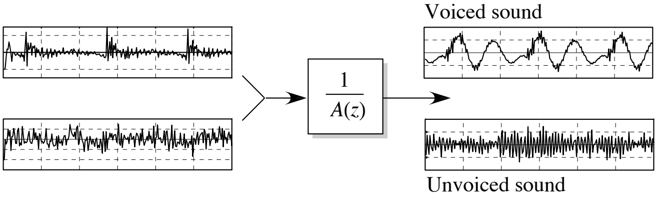


Figure 12.13 – *Creating voiced and unvoiced sounds with all pole filtering*

Exercise 12.4 (Compression of a speech signal)

The task you are supposed to do is divided in three parts:

1. Detecting whether a sound is voiced or unvoiced, and pitch measurement: when a signal is periodic, its autocorrelation function, defined by the normalized covariance (see definition 8.4 on page 275), shows maxima distant from each other by one period of the fundamental. Hence the idea to detect whether or not the sound is voiced and to measure the pitch based on the computation of the estimate of the autocorrelation function:

$$J(k) = \frac{\sum x(n)x(n-k)}{(\sum x^2(n))^{1/2}(\sum x^2(n-k))^{1/2}} \tag{12.12}$$

The use of this function, incidentally, will be justified by a least squares approach when we deal with the similar problem of cardiac rhythm estimation (see page 494).

As we have seen, the pitch belongs, in practice, to an interval $(F_{\text{min}}, F_{\text{max}})$. You can therefore restrain the computation of the function $J(k)$ to the values of k greater than $k_{\text{min}} = F_s/F_{\text{max}}$ and smaller than $k_{\text{max}} = F_s/F_{\text{min}}$, where F_s refers to the sampling frequency. If the maximum of $J(k)$ goes beyond a certain threshold, typically 0.6, the sound is considered voiced, and the pitch period is then given by the value k_0 corresponding to the maximum of $J(k)$.

Write a function that detects sound activity, determines whether the sound is voiced or unvoiced and measures, in the case of a voiced sound, the pitch period. Use this function to divide the signal voiced and unvoiced areas, using windows of 240 points, that is 30 ms of signal, with an overlap rate of 25%.

2. All pole filter parameter extraction based on the previous partitioning:
   - Use the `xtoa` function from page 330 to extract the P coefficients of the all pole filter associated with each analysis window. You can choose $P = 20$ as the model order for a voiced sound window and $P = 10$ for an unvoiced sound window.
   - Create a coefficient file containing for each block the coefficient of the all pole model, as well as the value of the pitch when the signal is voiced. Set the pitch value to zero to indicate an unvoiced sound.
3. Synthesis: write a program that achieves the synthesis. To tone down the possible sudden variations from the coefficients of one window to those of the next, set the overlap to 25% for the consecutive outputs calculated over time intervals of 30 ms. Use a white Gaussian noise or a simple sequence of impulse with identical amplitudes as the filter's input, depending on whether the sound is voiced or unvoiced.

In exercise 12.4, we did not quite perform the encoding operation we said we would. What we should have done is use 176 bits to encode all 22 parameters for every time of analysis, but that is not as simple as it seems, because we cannot just simply associate 8 bits to each of the 22 parameters, the bit distribution has to be optimal. A classic approach is to study, using a speech database, the parameter distribution and to create a codebook of representative elements by a vector quantization (see paragraph 12.15.2).

A *voice activity* detector usually precedes the whole encoding system. Its role is to determine the segments containing "silence", for which no parameter estimation is done. When reconstructing the signal, a faint noise is added to these segments of silence for purposes of listening comfort.

12.4 DTW

In signal processing, and particularly in the field of speech, different series of measurement conducted in seemingly identical conditions can provide recordings that actually show significant differences in terms of amplitude, duration, utterance speed, etc. The algorithm, which will be detailed later, performs a time-alignment of two observation sequences independently from possible differences in amplitude, duration, or utterance speed. It is commonly called the *DTW* algorithm, short for *dynamic time warping.*

In 1975, Itakura [34] suggested using the DTW for speech recognition. In the particular case of the recognition of isolated words, a dictionary is used, containing the sound recordings of the words to be recognized, and during the recognition operation, it has to be decided which word has been pronounced based on the observed sound signal. If the sound signal corresponding to a given word were perfectly reproduced, we would simply have to subtract, sample by sample, the sample signal from the reference signal: a word would then be recognized if the difference is equal to zero. Unfortunately, this is never the case. So, in order to perform the comparison, we are going to associate as best we can the consecutive phases of the signal to be recognized with those of the different dictionary signals and find a value for the discrepancy. The DTW performs both those operations.

The DTW algorithm

Consider two sequences of length d vectors $\{\mathbf{x}_1, \dots, \mathbf{x}_I\}$ and $\{\mathbf{y}_1, \dots, \mathbf{y}_J\}$. Let:

$$d(i,j) = \sqrt{\sum_{\ell=1}^{d}(x_{i,\ell} - y_{j,\ell})^2}$$

with $1 \leq i \leq I$ and $1 \leq j \leq J$, be the distance between the two vectors $\mathbf{x}_i$ and $\mathbf{y}_j$.

A solution to the time alignment problem consists of taking one by one the indices of the sequences $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_j\}$ using a pair of functions $\phi = (\phi_x, \phi_y)$ defined on $\{1, \dots, T\}$ and within the range $\{1, \dots, I\} \times \{1, \dots, J\}$, and to calculate the cumulated sum of the distances associated with ϕ:

$$d_\phi(I,J) = M_\phi^{-1} \sum_{k=1}^{T} m_k d(\phi_x(k), \phi_y(k))$$

where the m_k are positive weighting coefficients and M_ϕ is a normalization constant given by:

$$M_\phi = \sum_{k=1}^{T} m_k$$

It seems M_ϕ should depend on the choice of ϕ. In practice, the m_k are chosen such that M_ϕ is independent of ϕ. This can be achieved for example by choosing:

$$m_k = (\phi_x(k) - \phi_x(k-1)) + (\phi_y(k) - \phi_y(k-1)) \tag{12.13}$$

The function ϕ satisfies different types of constraints, such as:

- initial and final values: $\phi_x(1) = 1$, $\phi_x(T) = I$, $\phi_y(1) = 1$, $\phi_y(T) = J$;
- trajectory monotony and continuity: $0 \leq \phi_x(k) - \phi_x(k-1) \leq 1$ and $0 \leq \phi_y(k) - \phi_y(k-1) \leq 1$;
- local continuity: finally, the pair $(\phi_x(k), \phi_y(k))$ satisfies certain pathfinding rules.

Examples of pathfinding rules

To ensure local continuity, a sequence of possible paths is defined by a graph such as the ones shown in figures 12.14 and 12.15: the arrows in these figures show the only possible paths that can be taken to reach the final point.

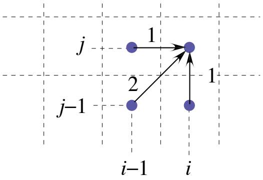


Figure 12.14 – *Pathfinding constraints: the numbers indicate the weight associated with the considered paths*

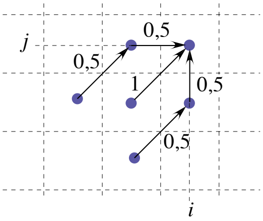


Figure 12.15 – *Pathfinding constraints: the numbers indicate the weight associated with the considered paths*

The goal of the DTW algorithm is to determine:

$$D(I,J) = \min_{\phi \in \Phi} d_\phi(I,J)$$

where Φ is the set of functions that satisfy the constraint.

Dynamic programming is a recursive approach which allows the previous criterion to be minimized using the following property:

Consider two sequences $\{\mathbf{x}_1, \dots, \mathbf{x}_I\}$ and $\{\mathbf{y}_1, \dots, \mathbf{y}_J\}$ and let $\mathcal{C}(I, J)$ be the minimal length DTW path associated with these two sequences. Then the sub-path of the path $\mathcal{C}(I, J)$ which reaches the point with coordinates $(\mathbf{x}_i, \mathbf{y}_j)$ is optimal for the two sub-sequences $\{\mathbf{x}_1, \dots, \mathbf{x}_i\}$ and $\{\mathbf{y}_1, \dots, \mathbf{y}_i\}$, because among all the paths that lead to the point with coordinates (i, j), minimization means we only keep the shortest one and therefore:

$$D(i,j) = \min_{\phi \in \Phi} d_\phi(i,j)$$

Thus, for the constraint graph shown in Figure 12.14, we infer that:

$$D(i,j) = \min \left\{ \begin{array}{l} D(i-1,j) + d(i,j) \\ D(i-1,j-1) + 2d(i,j) \\ D(i,j-1) + d(i,j) \end{array} \right\} \tag{12.14}$$

Likewise, for the constraint graph shown in Figure 12.15, we have:

$$D(i,j) = \min \left\{ \begin{array}{l} D(i-2,j-1) + 0.5d(i-1,j) + 0.5d(i,j) \\ D(i-1,j-1) + d(i,j) \\ D(i-1,j-2) + 0.5d(i,j-1) + 0.5d(i,j) \end{array} \right\} \tag{12.15}$$

Exercise 12.5 (DTW) Write a function that implements the DTW algorithm by taking the pathfinding constraints found in the graph in Figure 12.14. Use the two sequences of vectors as the input and as the output the array of the cumulated sums $D(i, j)$, as well as the value $D(I, J)/(I + J)$ as the output.

Exercise 12.6 uses the function obtained in exercise 12.5 to recognize a word in a given list of words. We will first briefly describe the cepstral analysis used to provide the characteristic sound features the DTW alignment is based on.

Cepstral coefficients

We know that the phonetic content of a speech signal can be characterized in a satisfactory way by a short-term spectral representation. This is why in speech recognition the first signal is usually performed, consisting of such an analysis over windows of about 20 milliseconds with overlap. Most of the time, the extracted characteristics are the first values of the short-term *cepstrum*, defined as the inverse Fourier transform of the logarithm of the modulus of the signal's DFT. If $x(n)$,with $n \in \{0, \dots, N-1\}$, refers to a portion of a signal with a duration of N, possibly weighted by a window, then the cepstral coefficients have the expression:

$$c(k) = \frac{1}{L}\sum_{\ell=0}^{L-1} S(\ell)e^{2j\pi n\ell/L} \text{ with } S(\ell) = \log\left|\sum_{n=0}^{N-1} x(n)e^{-2j\pi n\ell/L}\right|$$

It can be proven that the first cepstral coefficient represents the energy of the signal segment and that the following d coefficients, where d corresponds to a duration of a few milliseconds, characterize the shape of the vocal tract. In speech recognition, the relevant information is essentially characterized by the shape of the vocal tract. This is why most recognition methods are based on the use of the first cepstral coefficients. Most of the time, the cepstral coefficients are calculated on a logarithmic frequency scale; this is called *MFCC*, for *Mel Frequency Cepstral Coefficients.* Exercise 12.6 only uses a linear frequency scale, leading to a sequence of length d vectors that will undergo the recognition processes.

Exercise 12.6 (DTW word recognition)

1. Write a function that extract the short-term cepstrum from a speech signal sampled at 8,000 Hz using the operations:
   - multiplication of the time window by a Hamming window: choose a window duration corresponding to 20 ms with a temporal overlap of 50%;
   - computation of the logarithm of the modulus of the Fourier transform over $L = 256$ points;
   - computation of the inverse Fourier transform;
   - extraction of the 12 values useful to recognition.
2. Write a program that uses the DTW to compute, based on the sequences of cesptral coefficients, the "distance" between two recordings of the same pronounced word, and between two recordings of two different pronounced words.

12.5 Modifying the duration of an audio signal

Modifying the temporal scale of a sound has applications in many fields, such as solutions for the hearing impaired, speech design and recognition, movies, TV and radio advertisement, etc. A simple way of performing this modification is to reconstruct the signal with a sampling frequency different from the one used for the acquisition. Unfortunately, this method causes frequency distortion, because, as we know, if $X(f)$ represents the Fourier transform of $x(t)$, then the Fourier transform of $x(\gamma t)$, with $\gamma > 0$, is given by $\gamma^{-1}X(f/\gamma)$. Therefore, the frequency axis is dilated or contracted, depending on whether γ is greater or smaller than 1. You can observe the effects on a speech signal with the use of the MATLAB® function `sound` and by trying different reconstruction frequencies: when the frequency is greater than the acquisition frequency, the pitch seems higher. To prevent this type of distortion, several techniques have been suggested. The most popular one is probably the technique called *PSOLA*, for *Pitch Synchronous Overlapping Addition*, which works in the time domain [67]. Another one is referred to as the "phase vocoder", which works in the frequency domain [36]. One of the drawbacks of PSOLA is to add unwanted noises. As for the phase vocoder, it causes a reverberation effect.

12.5.1 PSOLA

To reduce the total duration of the signal, while preserving the frequency scale, we can simply eliminate small segments of the signal throughout the recording. Conversely, to increase the total duration of the signal, we can duplicate some of its segments. However, these segments have to be long enough to prevent spectrum aliasing, but short enough compared with the duration of the basic acoustic units: with PSOLA [67], the durations of the segments have to be chosen equal to the "instantaneous" pitch period. PSOLA is essentially comprised of two steps:

- **Analysis**: the signal $s(t)$ is time-"marked" according to a sequence of analysis times $t_a(i)$ such that $t_a(i) = t_a(i-1) + P_a$, where P_a refers to the "instantaneous" fundamental period (the pitch) estimated with a window long enough, starting at the time $t_a(i-1)$.
- **Synthesis**: the modified signal $\tilde{s}(t)$ is constructed by an overlap-add operation on the basic segments $s_i(t)$ of the original signal *relocated* at synthesis times $t_s(j)$ according to the expression:

  $$\tilde{s}(t) = \sum_n \tilde{s}_i(t - t_s(n))$$

  where $\tilde{s}_i(t - t_s(n)) = h(t - t_a(i(n)))s(t)$ is a signal segment centered in $t_a(i(n))$. The synthesis times are such that $t_s(n) = t_s(n-1) + P_a(i(n))$. If γ refers to the speed modification rate, the index $i(n)$ is the closest integer to the value $n\gamma$.

Exercise 12.7 (PSOLA)

1. **Analysis**: use the `f0cor.m` function, which estimates a signal's pitch, to construct the sequence of analysis times $t_a(n)$ in the following way:

- The initial values are set as $t_a(1) = 1$ and $P_a = L_{10}$ where L_{10} is the number of samples corresponding to 10 ms.
- The window starting at the time $t_a(n)$ and lasting at least two pitch periods P_a is extracted. In practice, the duration has to be chosen equal to twice the smallest period expected in the signal.
- Pitch detection is performed on the window. If the signal is voiced, we get the pitch period P_a and the analysis time $t_a(n+1) = t_a(n) + P_a$. If the signal is unvoiced, the value $t_a(n+1) = t_a(n) + L_{10}$ is chosen.

At the end of this first process, we have a sequence of N_a analysis times (positions in the file) $t_a(n)$ that are *synchronous* with the pitch period.

2. **Synthesis**: we wish to modify the prosodic speed by a factor γ. This is done by generating the sequence of synthesis times t_s as follows:

- Initially, $t_s = 1$.
- The following synthesis time is calculated by executing:

```
te=te+gamma;
ie=ceil(te);
```

The index is used to point out the analysis time after which the segment of the signal used for creating the desired signal is extracted. Hence, if $\gamma < 1$, which corresponds to a slower utterance, the value of i_e after n iterations will be smaller than n. Hence, some segments of the signal will be repeated and the signal created will last longer.

- The sequence of synthesis times is generated by executing:

```
ts=ts+(ta(ie+1)-ta(ie));
```

This is equivalent to generating a window with a length equal to the pitch (hence the phrase Pitch Synchronous), because (using obvious notations), we can write:

$$t_s(n+1) = t_s(n) + \underbrace{(t_a(i_e(n)+1) - t_a(i_e(n)))}_{P_a}$$

Array 2 shows some of the values for the sequences that were found for $\gamma = 0.8$. As you can see, the portion centered in 7,668 in the original signal is in position 9,610 in the synthesis signal. Notice also that, because the signal is slowed down, as we wanted it to be, we have to duplicate the portion centered un 7,629, once in position 9,532, and once more in the following position 9,571. The signal synthesis is performed as follows:

- the segment centered in $t_a(i_e)$ and with a length of $2(t_a(i_e+1) - t_a(i_e))$ is extracted from the original signal;

n	1	...	199	200	201	202	...
$t_e(n)$	1	...	159.4	160.2	161.0	161.8	...
$i_e(n)$	1	...	160	161	161	162	...
$t_s(n)$	1	...	9,493	9,532	9,571	9,610	...
$t_a(i_e(n))$	1	...	7,590	7,629	7,629	7,668	...

Table 12.2 – Sequences of synthesis and analysis times corresponding to $\gamma = 0.8$, that is a slower utterance

– the extracted segment is multiplied by a Hann window then added to the previous portion with a 50 % overlap.

Write a function that consecutively performs the two steps of the process.

12.5.2 Phase vocoder

The basic idea behind the *phase vocoder* [36] is to perform a short-term Fourier analysis. If the spectrum is comprised of narrow band spectral components, which means that the sound is closer to a voiced sound than it is to an unvoiced sound, and that the analysis window is much longer than the pitch period, then the values of every pitch harmonic will be clearly identified. In this case, if the spectral characteristics are maintained, for a duration slightly lower or slightly greater than the original one, then the value of the pitch is preserved.

The only difficulty is that the phases associated with each frequency component in the modified signal's spectrum have to be calculated in such a way as to ensure the proper alignment of the consecutive phases during the overlap-add operation.

Exercise 12.8 (Hann window)
Consider the sequence (called a *Hann window*):

$$h(n) = \begin{cases} 0.5(\cos(2\pi n/L) + 1) = \sin^2(\pi n/L) \text{ for } n \in \{0, \dots, L-1\} \\ 0 \text{ otherwise} \end{cases}$$

Let $\alpha \in (0, 1)$, and [2] $n_0 = \lfloor \alpha L \rfloor$. Write a program that plots against n the sequence:

$$x(n) = \sum_k h^2(n - kn_0) \tag{12.16}$$

Notice that the sequence $h(n)$ has a finite length. Therefore, to construct $x(n)$, all you need to do is to calculate the sum of a finite number of segments distant from each other by $h^2(n)$.

[2] $\lfloor \alpha L \rfloor$ refers to the integer part of αL.

What happens when L varies, when α varies? Try other powers of $h(n)$ in expression 12.16.

Exercise 12.9 (Phase vocoder)
Write a program that performs the following operation:

- Calculation of the DFTs of the signal segments, each segment weighted by a length L Hann window $h(n)$. The consecutive windows are distant from each other by a number of points $n_0 = \lfloor \alpha L \rfloor$ where $\alpha \in (0,1)$. With such an overlap, we have:

  $$\sum_k g(n - kn_0) = C$$

  where $g(n) = h^2(n)$ and where C is constant depending on α that you will determine. According to what we saw in exercise 12.8, if L is a power of 2, the distance between the windows has to be in the form $n_0 = L/2^m$.

- Generating the sequence of synthesis times with a factor γ compared with the original sequence according to the method suggested in exercise 12.7.

- Calculation of the modified DFTs by performing an amplitude interpolation, proportional to the original DFTs on the interval containing the synthesis time. For the phase calculations, sum the phase increases of the original DFTs.

- Calculation of the inverse DFTs of the modified DFTs, each DFT being once more weighted by a length L Hann window. For $\gamma = 1$, everything happens as if the window $g(n) = h^2(n)$ were applied and, therefore, the correct amplitudes can theoretically be found by dividing the obtained signal by the constant C.

12.6 Quantization noise shaping

Shannon [88] showed that, for a given level of distortion, there is an encoding that minimizes the *rate* measured in bits per second (bps). The first problem is to define a measurement of distortion. We saw one possible definition in the example on the signal-to-quantization noise ratio. A second difficulty resides in the fact that, in practice, the relation between distortion and minimal rate is usually impossible to determine, because it would require an analytical model for the signal statistics, which most of the time we don't have. Last but not least, the theorem does not tell us how to reach the fundamental limit.

Acquisition

Audio frequency signals are usually real, B band signals. The signal is sampled at a frequency $F_s \geq 2B$, then its samples are quantized, by linear quantization using N bits. The rate is then equal to $F_s \times N$ bps. To increase the SNR, we can therefore increase N during the acquisition; remember that if $F_s = 2B$, the SNR increases by about 6 dB per quantization bit. On the other hand, we can also increase F_s beyond the Nyquist frequency $2B$. One solution (Figure 12.16) to take advantage of this oversampling consists of rejecting part of the quantization noise's power outside the useful band $(-B, +B)$. This noise shaping operation was first introduced by Cutler in 1954.

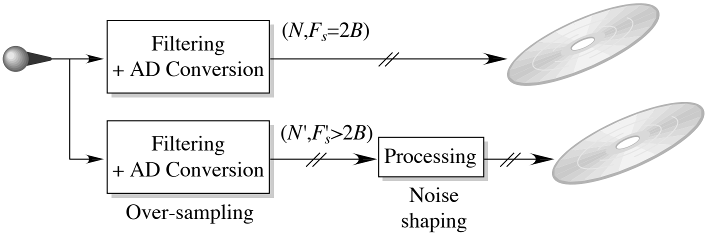


Figure 12.16 – *Acquisition for an audio CD with and without oversampling*

Generally speaking, in order to decrease the rate for a given level of distortion, the signal statistics have to be taken into account, but also the way the signal is perceived. This is what has been implemented in certain compression techniques where the samples are replaced by quantities, such as the prediction coefficients in the case of speech, or by the DCT coefficients in the case of music.

Concerning the sampling of $B = 22$ kHz band audio signals, the compromise between sound quality and technical feasibility has led to several formats, the most common of which are:

– the audio CD, which uses a sampling frequency of $F_s = 44.1$ kHz and an $N = 16$ bits quantization. The resulting rate is roughly 706 kbps, with an SNR of about 96 dB;

– the audio DVD, which uses a sampling frequency up to $F_s = 192$ kHz and a quantization up to $N = 24$ bits. This corresponds to a rate of 4.6 Mbps and an SNR of about 144 dB;

– the DSD (Direct Stream Digital) which performs a *one* bit quantization at a sampling frequency $F_s = 128 \times B$, hence a rate of 2.8224 Mbps. The resulting SNR is approximately 120 dB. Notice that the SNR is the same as the one resulting from a 20 bit format sampled at 44.1 kHz, which corresponds a third of the rate. The DSD is usually chosen for technological reasons.

Reconstruction

The noise shaping operations mentioned previously can also be used for the reconstruction (Figure 12.17). The first operation consists of "increasing" the sampling frequency: this actually an *interpolation* which is sometimes inaccurately called oversampling. This interpolation is followed by quantization noise shaping. Here are a few implementation examples:

- Probably one of the oldest systems was the one imagined by Philips in audio CD players to recover 16 bit quality at 44.1 kHz while using a 14 bits analog-to-digital converter, but set to four times the initial frequency, hence $4 \times 44.1 = 176.4$ kHz.
- The more recent *MASH* digital-to-analog converter (for *Multi-stAge noise-SHaping*), introduced by the Nippon Telephone and Telegraph, operates on 4bits at a frequency of about 3 MHz. It is used in certain audio DVD players.

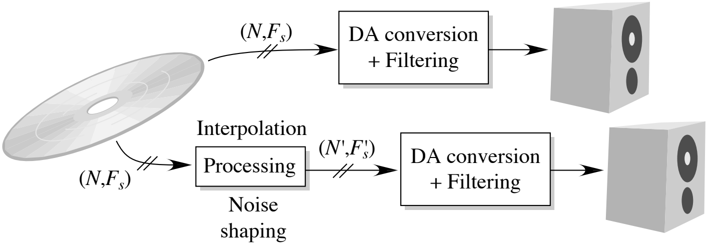


Figure 12.17 – *Reconstruction of audio CD signal, with and without processing*

- Finally, we will mention the One-Bit Stream technique introduced by Philips for audio CDs: the digital-to-analog converter only has a *one* bit resolution but operates at an interpolation frequency between 128 and 256 times the Nyquist frequency. The audio CD's initial quality can be restored by quantization noise shaping and a post-filtering.

Notice that in the case of the DSD format, it is possible to go back to the 20 bit format at the frequency 44.1 kHz with the initial 120 dB quality. However, this requires very long filters to suppress the frequencies outside the useful band. The DSD can be seen as some kind of generic format, based on which an entire range of other formats can be defined with varying levels of quality.

The following exercise is meant to illustrate these interpolation and quantization noise shaping techniques.

Exercise 12.10 (Spectral quantization noise shaping)
In this exercise, the signal $x(t)$ is the sum of three sines with the respective frequencies 437, 504 and 1,367 Hz. The following program generates samples of $x(t)$ for the sampling frequencies 44,100 Hz (SE1) and 4 × 44,100 Hz (SE2):

```
%===== MISFQ.M:
% Signal generation
clear; surech=4;            % Oversampling factor
AA=[1.2 3.2 2.7];           % 3 amplitudes
freq=[437 504 1367]'/44100; % 3 frequencies
T=200;
SE1=AA*cos(2*pi*freq*(0:T-1));
SE2=AA*cos(2*pi*freq*(0:surech*T-1)/surech);
```

1. In order for the errors introduced by the quantization to be seen, they have to be much greater than the errors introduced by the interpolation operations. The object of this first question is to justify the choices that will be made concerning the number of bits used to encode in order to perform a simulation that shows the properties of the quantization.

   Using the the interpolation function[3] obtained in exercise 4.14 on page 152, calculate the sequence SE2int corresponding to the frequency $f_{e2} = 4 \times f_{e1}$. Create a program that compares SE2int with the correct sequence SE2. In order to do this, calculate the ratio of the signal's root mean square value to the difference (SE2-SE2int) using the MATLAB® function std. Perform this computation on a reduced range so as to avoid the side effects caused by the interpolation function.

2. The samples taken at the frequency $f_{e1} = 44{,}100$ Hz are $N_1 = 8$ bit quantized. Let SE1Q1 be the resulting sequence. Write a program that evaluates the signal-to-quantization noise ratio for the signal. Formula 7.34 indicates that you should have roughly 48 dB.

   Same question with $N_2 = 6$ bits, where SE1Q2 is the resulting sequence. Check that there is a loss of about 12 dB.

3. The sequence SE1Q1 is interpolated at the frequency $f_{e2} = 4 \times f_{e1}$. This leads to the sequence SE2Q2. It is important that you notice that the sequence SE2Q2 contains quantization noise outside the frequency band of the signal $x(t)$, and therefore that SE2Q2 has to be filtered before comparing the result with SE2. This is done by using the rif function obtained in exercise 4.8. Based on the expression of the quantization noise's power, show that there is a gain of about 6 dB. Check this result with a simulation.

[3] Those who have access to the Signal Processing Toolbox can also use the interp function, which at the cost of an algorithm which is a bit more complicated leads to a slightly better interpolation.

4. Consider the process represented by the diagram in Figure 12.18 where Q is a system that goes from 8 bits to 6 bits by eliminating the two least significant bits.

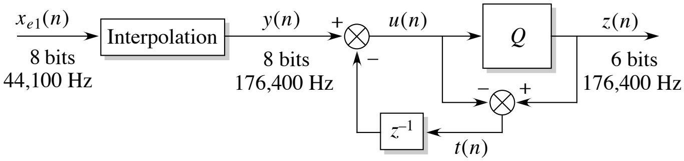


Figure 12.18 – *Noise shaping*

Therefore, it is equivalent, based on the diagram in Figure 7.8 on page 271, to adding a white noise $e(n)$ with the power spectral density equal to $q^2/12$ in the band.

Use this result to show that $z(n) = y(n) + e(n) - e(n-1)$. By noticing that going from $e(n)$ to $\varepsilon(n) = e(n) - e(n-1)$ is a linear filtering, infer that the signal-to-noise ratio is enhanced by about 6 dB compared with a system that would directly apply the quantization to $y(n)$.

This is an interesting result, from a practical point of view, since it allows you to find basically the same SNR as in question 1. Notice that the suggested process cannot be done without the interpolation operation conducted on the quantized signal with many more bits than what is necessary, in this case 8 at the beginning compared with the 6 at the end.

12.7 Elimination of the background noise in audio

For practically every signal processing system, we are faced with the problem of extracting a useful signal corrupted by noise. In the case of audio frequencies, there are several different causes for the noise. We will mention for example the noise due to the defects in the sensors and, in other recording devices, the noise related to the environment in which the recording was done (conversation noises in a public place, doors opening and closing, the tinkling of silverware in a restaurant, etc.) or also the decay of the recording medium. Among all the types of noises encountered in practice the following two types occur in almost all audio systems:

– The *background noise* which can be represented by a stationary random process. It often originates from the electronic circuits of the amplification and reconstruction devices, hence the name.

– The *impulse noises* which can be represented as a sequence of very brief impulses, with random amplitudes and locations. Listening to a scratched record is a perfect example.

To reduce these noises and restore the recording as best as possible, we are going to consider them and deal with them separately.

Eliminating background noise is a particularly difficult problem, not to say impossible to solve, since we have to find what characterizes the difference between the useful part of the signal and the unwanted part. Take the example of a musical recording where we can hear the sound of a violin amidst background noise. The program assigned with the separation would have to transform into some kind of a music expert to distinguish the sound of the violin from the background noise. As a consequence, the solution can sometimes be worse than the problem when reducing background noise in musical recordings.

If we only concern ourselves with speech signals, we can mention [32, 33, 11] among the different denoising methods in a stationary background noise. These methods give satisfactory results, such as for example in cellular technology, or for restoring "old" recordings, but only if the settings of the algorithm are very well chosen. They usually require for the spectral properties of the noise to be known. They can do this by using a portion of the signal that contains nothing but noise. This portion is selected either by a direct "acoustic" observation of the signal, or by automatically detecting the active areas of the useful signal. In [11], the author suggests a very simple method leading to an acceptable result that we are going to implement. If we assume that the noise is white with the power σ^2, known or estimated from a portion of the signal containing nothing but noise, the short-term amplitude spectrum of the denoised signal σ^2 is estimated by the expression:

$$\widehat{X}(k) = G(k)X(k) \tag{12.17}$$

where:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-2j\pi nk/N}$$

refers to the DFT of a length N window of the noisy signal $x(n)$ and where:

$$G(k) = \begin{cases} \left(1 - \lambda\dfrac{\sigma\sqrt{N}}{|X(k)|}\right) & \text{if } \lambda\dfrac{\sigma\sqrt{N}}{|X(k)|} < 1 \\ \mu & \text{otherwise} \end{cases}$$

where the parameters λ and μ are adjusted experimentally so as to obtain the best sound result. This method is sometimes called spectral subtraction. According to 12.17, everything happens as if we were applying a gain $G(k)$ adapted to each component of the DFT. The results show that the background noise is rather well eliminated, but the method introduces whistling noises, that get louder as μ decreases. In particular, the value $\mu = 0$ leads to the plain and simple elimination of the spectral components below the threshold, causing certain components of the noise to be isolated and to behave as isolated "peaks". This "whistling" noise, referred to as *musical noise*, can be reduced by decreasing λ and increasing μ, but at the cost of a lower noise reduction.

Exercise 12.11 (Denoising a speech signal)
First, record a speech signal, sampled at 8,000 Hz, which will serve as the reference signal. Construct a noisy version of it, by adding noise so as to ensure a signal-to-noise ratio of 10 dB.

Implement the method described by equation 12.17 to denoise the noisy file by considering that σ is known. Try the program for different values of N, of λ and of μ.

12.8 Eliminating the impulse noise

We are now going to look into the restoration of recordings containing errors with a relatively large amplitude and with a very brief duration (less than a millisecond), referred to as *clicks*, and assumed to be in small numbers. Thus, for the signal represented in Figure 12.19, originating from record, shows a scratch spread over a few samples. Cracks are not always as "visible" as this one, and the recording often has to be listened to in order to detect them.

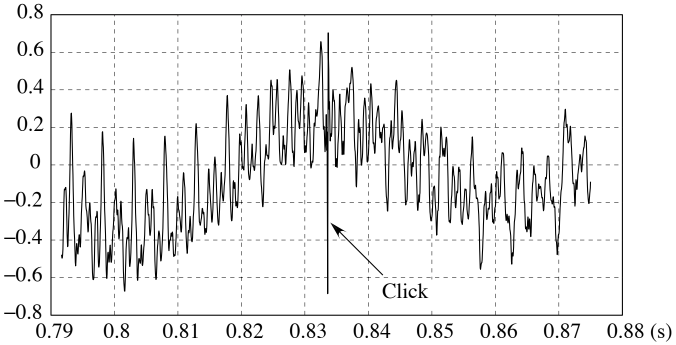


Figure 12.19 – *Signal originating from a record and containing a click*

The considered restoration method is comprised of two separate steps:

1. the detection of clicks in the signal;
2. the restoration of corrupted samples.

12.8.1 The signal model

When observing the signal $s(n)$ shown in Figure 12.19, what makes us decide that there is a click in position n_0 is that the value in n_0 is noticeably different from what would seem *predictable* to us based on the past signal. Therefore, if we know how to calculate a prediction $\hat{s}(n)$ based on the last K values $s(n-1), \ldots, s(n-K)$, the procedure can be made automatic, by deciding the presence of a click in position n if the difference between $\hat{s}(n)$ and $s(n)$ is above a threshold that can be determined experimentally.

We showed in Chapter 11 that, for an WSS process, the best K order linear predictor:

$$\hat{s}(n) = \alpha_1 s(n-1) + \cdots + \alpha_K s(n-K)$$

in the least squares sense was obtained by choosing the solutions to the Yule-Walker equations 11.28 as the coefficients α_i. We have also determined, with equation 11.27, the expression of the prediction error σ^2. Remember that according to property 11.4, if $s(n)$ is a K order AR process, the prediction process $s(n) - \hat{s}(n)$, also called the residual signal, is *white*. Because we are going to use this property in click detection, we will assume that this is the case: *in the absence of clicks, the signal $s(n)$ is a K order AR process.* From now on, K is assumed to be known. In practice, its value is set by examining the results.

If the number of clicks is small, we can assume that the *effects caused by their presence in the estimation window are negligible.* We can then estimate the parameters $a_1, \ldots, a_K$ and σ^2 of the K order AR process directly on the signal's window. Once these values have been estimated, the FIR filter with the transfer function $A(z) = 1 + a_1 z^{-1} + \cdots + a_K z^{-K}$ can be applied to the signal $x(n)$. The signal $y(n)$ is obtained. Because $s(n)$ is a K order AR, then in the absence of clicks, this signal $y(n)$ is a white noise with the variance σ^2. If, in addition to that, $s(n)$ is Gaussian, then $y(n)$ is itself Gaussian. Let us now see how to detect the presence of clicks.

12.8.2 Click detection

We now assume that a click is described as an impulse $i(n) = A_0\delta(n-n_0)$ added to the useful sound signal $s(n)$. The observed signal is then $x(n) = s(n) + i(n)$. We are going to try to detect the possible presence of $i(n)$ using a linear filter, then by comparing the filter's output with a threshold. The following exercise shows how the methods works.

Exercise 12.12 (Detecting impulse clicks)
Let $d(n)$ a signal with a known shape, corrupted by a noise $b(n)$. "Detecting" the signal $d(n)$ means we have to choose a decision rule to be able to say if the signal $d(n)$ is or not in the observed signal $y(n)$. Hence we have two hypotheses that can be summed up as follows:

– in the absence of the signal, what we see is $y(n) = b(n)$;

– in the presence of the signal $d(n)$, what we see is $y(n) = d(n) + b(n)$.

We will assume that $b(n)$ is a white noise, with the variance σ^2. To conduct the processing, we are going to impose that the detector is comprised of a linear filter followed by a threshold comparator, and we will find the optimal settings for the the filter and the threshold value.

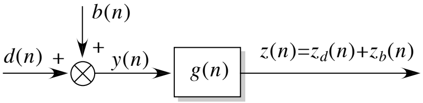


Figure 12.20 – *Matched filter*

Let $z_d(n)$ and $z_b(n)$ be the outputs of the filter $g(n)$ we are trying to determine, the inputs of which are the deterministic signal $d(n)$ and the noise $b(n)$.

1. Show that the signal-to-noise ratio defined by $\rho = |z_d(n)|^2/\mathbb{E}\{|z_b(n)|^2\}$ has the expression:

$$\rho = \frac{1}{\sigma^2}\frac{\left|\sum_{u=-\infty}^{+\infty} g(u)d(n-u)\right|^2}{\sum_{u=-\infty}^{+\infty} |g(u)|^2}$$

2. Using the Schwartz inequality, find the expression of the impulse response of the filter $g(n)$ that maximizes the value of ρ. In the literature, this filter is called a *matched filter* (the phrase implies that the filter is matched with the signal $d(n)$).

3. Consider now the problem of detecting, with a linear filter $h(n)$, an impulse $\delta(n)$ in a K order AR signal defined by $s(n) + a_1 s(n-1) + \cdots + a_K s(n-K) = w(n)$ where $w(n)$ is a white noise with the variance σ^2. To achieve such a detection, we can decompose $h(n)$ in a cascade of two linear filters $h_1(n)$ and $h_2(n)$, the first one a *FIR* filter with the impulse response $h_1(0) = 1$,$h_1(1) = a_1, \ldots, h_1(K) = a_K$ (Figure 12.21). Let $y(n)$ be this filter's output.

   By applying the result of the previous question, determine the filter $h_2(n)$ that maximizes the detection signal-to-noise ratio. Let $z(n)$ be the output of the filter $h_2(n)$.

4. In the absence of clicks, determine as a function of $a_1, \ldots, a_K$ and σ^2 the expression of the variance P_z at the output of the filter $h_2(n)$.

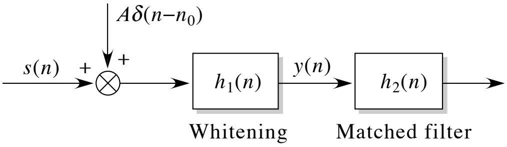


Figure 12.21 – *Signal whitening before detection*

5. A threshold s is set, and the following decision rule is adopted: if $|z(n)| > s$, the presence of a click is decided at the time n. Under the hypothesis that s is Gaussian, use the decision rule to show that the expression of the threshold that ensures a probability equal to α of deciding the presence of a click where there isn't one, is $s = \lambda(\alpha)\sqrt{P_z}$. For $\alpha = 0.01$, we have $\lambda = 3$.

6. Simulation: write a program that generates 500 samples of a 10 order AR process, simulating the useful signal $s(n)$. Use, as the parameters of the AR-10 process, the values $\sigma^2 = 1$ and $a_1, \ldots, a_{10}$ given by:

```
%===== AA.M
a= [1 -1.6507 0.6711 -0.1807 0.6130 -0.6085 0.3977 ...
    -0.6122 0.5412 0.1321 -0.2393];
```

   Let s_e be the root mean square value of $s(n)$, an estimate of which is given under MATLAB® by `seff=sqrt(s*s'/N)`. Add 5 clicks with amplitudes equal to ±$1.5s_e$ at arbitrary times. Let us assume that the order of the filter is known, but that, despite the presence of a few clicks, the model's parameters can be estimated with the `xtoa` function. Successively estimate the model's parameters, the whitening, the matched filtering, then the comparison with the previously determined threshold.

   The results obtained with the previous program show that a given click can lead to several close positions detected around the real value. In practice, it is preferable to "group" these positions together. Describe a processing algorithm corresponding to several positions detected for the same click.

SUMMING UP: the previous results lead us to summing up click detection in the following operations:

- estimation of the K order AR model's $K + 1$ parameters a_1, ..., a_K and σ^2 based on the length N window $x(n)$;
- filtering of $x(n)$ by the filter with the impulse response $\{1, a_1, \ldots, a_K\}$;
- filtering of the previous signal by the matched filter with impulse response $\{a_K, a_{K-1}, \ldots, a_1, 1\}$. The output signal is denoted by $z(n)$;

– comparison of $|z(n)|$ with the threshold:

$$s = \lambda\sqrt{\sigma^2(1 + a_1^2 + \cdots + a_K^2)}$$

where the parameter $\lambda \approx 3$ is set experimentally.

12.8.3 Restoration

Once a click is detected in position n_0, you would think that only one value has to be reconstructed: that of the altered sample. But in fact, the error is rarely found on only one point of the signal. This is why it is better to consider several samples on both sides of the detected position to be errors. This means we have a corrupt area of m consecutive values. Typically, m is chosen between 9 and 15 (see Figure 12.22).

The idea, which consists of performing a simple linear interpolation based on the values placed on either side of the corrupt area, does not take into account the correlations among the points of the signal. The solution we are going to use consists of using the prediction structure associated with equation 11.23 of a K order AR process.

Exercise 12.13 (Restoring “missing values”)
Let us assume the parameters of the AR model have been estimated, and that, in the considered block, the corrupt zone goes from position ℓ to position $\ell + m - 1$. We are going to try, by minimizing the square deviation, to search for the best values of $x(\ell), \ldots, x(\ell + m - 1)$ (Figure 12.22).

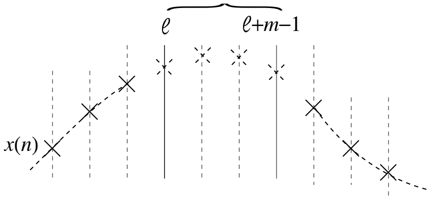


Figure 12.22 – *Several values are restored around the detected position*

1. Show that estimating the m unknown values can be seen as a linear problem depending on the non-corrupt values and the model’s coefficients.

2. Use this result to estimate, using the least squares method, $\widehat{\mathbf{y}}$ of the corrupt zone $\mathbf{y} = [x(\ell), \ldots, x(\ell + m - 1)]^T$.

   Write the expression of $\widehat{\mathbf{y}}$ in the form:

$$\widehat{\mathbf{y}} = -(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T(\mathbf{A}_0\mathbf{x}_0 + \mathbf{A}_1\mathbf{x}_1) \qquad (12.18)$$

where $\mathbf{A}_0$, $\mathbf{A}_1$ and $\mathbf{B}$ are matrices build from the model's coefficients $(a_1, \ldots, a_K)$ and $\mathbf{x}_0$ and $\mathbf{x}_1$ are vectors constructed from the non-corrupt observations $x_1, \ldots, x_{\ell-1}, x_{\ell+m}, \ldots, x_N$.

The plots in Figure 12.23 were obtained using the following program. They show the signal containing the clicks and the signal after restoration:

```
%===== RESTAU.M
% Signal reconstruction
% Run the detection program, then
% select a position to restore in the list of
% positions detected by detect.m
pos=input('Click position: ');
lsig=length(sig); tps=[0:lsig-1]; sig=reshape(sig,lsig,1);
m=15; ell=pos-7;
X0=sig(ell-K:ell-1); X1=sig(ell+m:ell+m+K-1);
colT=[aest(K);zeros(m+K-1,1)];
ligT=[aest(K:-1:1)' zeros(1,m+K)];
T=toeplitz(colT,ligT);
A0=T(:,1:K); B=T(:,K+1:K+m);
A1=T(:,K+m+1:2*K+m); X=A0*X0+A1*X1;
%===== Solving the system
Y=-B \ X; sigr=sig; sigr(ell:ell+m-1)=Y;
plot(tps,sig,'-r', tps,s,'b', tps, sigr,':y'); grid;
```

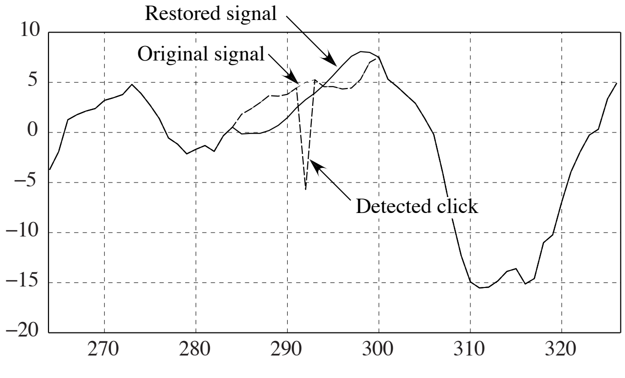


Figure 12.23 – *Clicks (dashed line), restored signal (full line)*

The method presented here is successfully used to clean up recordings, such as the ones stored on old records. It requires a few adjustments to determine experimentally the window size, the AR order and the threshold value that provide the best acoustic results.

12.9 Tracking the cardiac rhythm of the fetus

12.9.1 Objectives

The human heart's activity produces electrical currents that spread through the tissue and can be measured with the use of electrodes attached to the skin. These signals are called an electrocardiogram (ECG, or EKG). In obstetrics, these signals can be used to keep watch over the cardiac condition of the fetus, in particular during the delivery, which allows doctors to detect possible anomalies very early on, and hence to treat them more rapidly.

There are essentially two methods that are used to track the cardiac rhythm of the fetus. The first one, which can only be carried out during the delivery, consists of measuring the electrocardiogram (ECG or EKG) directly off the fetus by placing an electrode on its scalp. The second one, the main advantage of which is to be non-invasive, consists of placing electrodes on the mother to pick up signals from which the fetal cardiac signal will be extracted.

In the second method, the use of a single sensor placed on the mother's abdomen is not sufficient because the amplitude of the fetal EKG can be several times less than the noise produced by different sources of interference such as the mother's EKG, but also the signals caused by her muscle activity (electromyogram) or the ones related to her breathing. Therefore, the observation is difficult. However, the use of several sensors makes it possible to separate the signals (Figure 12.24).

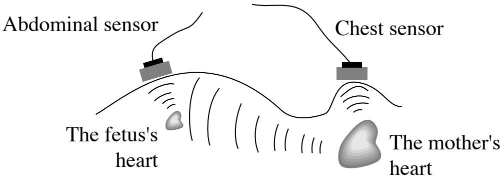


Figure 12.24 – *Picking up the EKGs*

The data we are going to process in this work come from measurements done on EKG signals samples at $F_s = 300$ Hz over a duration of a few seconds. Many websites make such signals available to the public. From now on, the signal originating from the sensor on the chest will be denoted by x_p and the one originating from the sensor on the abdomen by x_v (Figure 12.25).

A reading of such signals is shown in Figure 12.26.

Two processings have to be conducted. First, we have to extract the fetus's EKG from the signals $x_p(n)$ and $x_v(n)$ and, second, estimate the cardiac rhythm of the fetus based on the obtained signal.

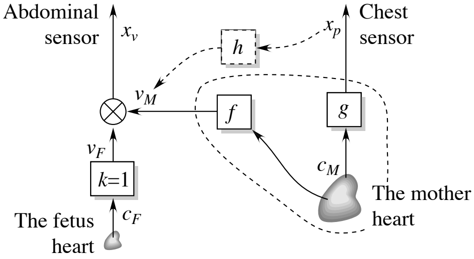


Figure 12.25 – *The model used for describing EKG tracking*

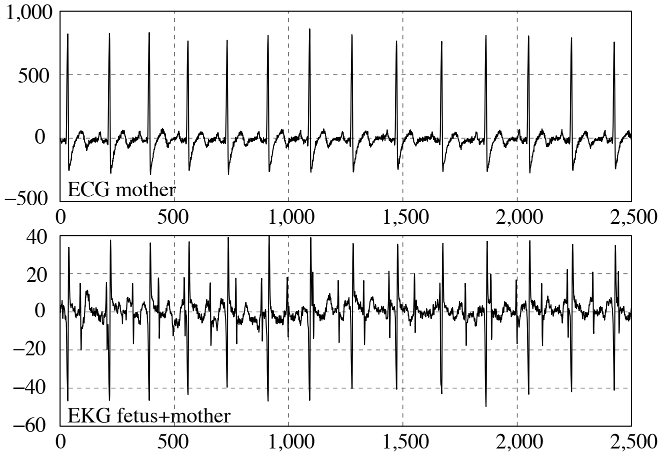


Figure 12.26 – *Mother-fetus EKG signals. Graph on the top: signal measured off the chest, and which can be considered as the signal originating from the mother's heart. Graph on the bottom: signal measured off the abdomen containing both cardiac signals*

12.9.2 Separating the EKG signals

Theoretically, the signal observed with the chest sensor represents the mother's cardiac signal noised by the fetus by the signal originating from the heart of the fetus. However, because this sensor is located far away from the heart of the fetus, because its heartbeat is faint, we will assume that the signal $x_p(n)$ represents the mother's heart filtered by traveling through the thoracic tissue. If the mother's cardiac signal is denoted by $c_M(n)$, and if traveling through the tissue acts as a linear filter with a length K_p finite impulse response, we have:

$$x_p(n) = g_1 c_M(n - n_p) + \cdots + g_{K_p} c_M(n - n_p - K_p + 1) \qquad (12.19)$$

where anyone n_p accounts for the overall propagation delay through the chest. In practice, this value turns out to be small.

As for the signal $x_v(n)$, it is the sum of the cardiac signal of the fetus, filtered by traveling through the abdomen tissue and a signal originating from the mother's heart. However, this last signal is not quite the signal $x_p(n)$ picked up by the chest sensor, because the signal produced by the mother's heart reaches the abdomen sensor after traveling through the abdomen. Therefore, it has undergone some transformations. By assuming once more that the abdominal transfer behaves like a linear filter with a length K_a finite impulse response filter, the signal originating from the mother's heart can be written (in fact the FIR hypothesis should be related to the comments made on the approximation of an IIR by a FIR):

$$v_M(n) = f_1 c_M(n - n_a) + \cdots + f_{K_a} c_M(n - n_a - K_a + 1)$$

where n_a accounts for the overall propagation delay through the abdomen. By replacing this expression in 12.19 we end up with a relation between the disruptive signal from the abdomen sensor and the signal from the chest sensor, which is written:

$$\begin{aligned} v_M(n) \quad &= \quad h_{-M} x_p(n+M) + \cdots + h_{-1} x_p(n+1) \\ &\quad\quad + h_0 x_p(n) + \cdots + h_{L-1} x_p(n-(L-1)) \end{aligned}$$

This expression calls for a few comments. The signal $x_p(n)$ is not exactly the one produced by the mother's heart, because the latter is subjected, according to 12.19, to a delay n_p due to the propagation through the thoracic tissue. Because the propagation times are unknown, n_p may very well be greater than n_a. In that case, $x_p(n)$ is delayed with respect to $v_M(n)$. This is why we planned for an "anticausal" part represented by the terms $h(-M)$,..., $h(-1)$. However, if $n_p < n_a$, which certainly is the case in this experiment, the signal $v_m(n)$ is delayed with respect to $x_p(n)$ and we have to make sure that the coefficients $h(-M)$, ..., $h(-1)$ are also almost null.

In the end, the two signals $x_p(n)$ and $x_v(n)$, picked up off the mother's chest and abdomen, are such that:

$$\begin{aligned} x_v(n) \quad &= \quad h_{-M} x_p(n+M) + \cdots + h_{-1} x_p(n+1) \\ &\quad\quad + h_0 x_p(n) + \cdots + h_{L-1} x_p(n-L+1) + c_F(n) \end{aligned}$$

where $c_F(n)$ represents the signal we are trying to determine, originating from the heart of the fetus. We are going to estimate the coefficients h_i in such a way as to extract $c_F(n)$ from the signals $x_p(n)$ and $x_v(n)$.

Example 12.4 (Extracting the signal $c_F(n)$)
The first part of our work consists of identifying the abdominal transfer:

1. Establish that estimating the linear sequence $\mathbf{h} = [h_{-M} \ \dots \ h_0 \ \dots \ h_{L-1}]$ is equivalent to a linear problem of the type:

   $$\mathbf{x}_v = \mathbf{X}_p \mathbf{h} + \mathbf{c}_F$$

   where $\mathbf{x}_v$ and $\mathbf{X}_p$ are respectively a vector and a matrix with the adequate sizes, constructed from the observations. $\mathbf{c}_F$ is a vector that represents the signal originating from the heart of the fetus.

2. Using the ordinary least squares method, find the expression of an estimate of $\mathbf{h}$. Use the result to find the estimate of the signal $c_F(n)$.

3. Based on the previous result, write a function `extract(Xp,Xv,M,L)` that estimates h_{-M}, ..., h_0, ..., h_{L-1} and infers an estimate of the signal $c_F(n)$ originating from the heart of the fetus.

COMMENT: the method considered for the extraction of the signal $c_F(n)$ assumes that the useful information concerning the signal $c_F(n)$ (such as its frequency, but also other characteristics useful to the practitioner) is not contained in the space generated by the columns of the matrix $\mathbf{X}_p$. The likelihood of this hypothesis can really be evaluated only by whether or not the results are relevant. If, for whatever reason, a part of the signal originating from the heart of the fetus actually does belong to the space generated by the columns of $\mathbf{X}_p$, then this signal is impossible to extract using the suggested method. If this signal happens to be useful to the practitioner, then another separation method will have to be considered. In this case, for the estimation of the cardiac frequency, the results obtained are quite satisfactory.

HINT:

1. The observed sequences are indexed from 1 to N. If we stack the expressions:

   $$\begin{aligned} x_v(n) &= h_{-M}x_p(n+M) + \cdots + h_{-1}x_p(n+1) \\ &\quad +h_0 x_p(n) + \cdots + h_{L-1}x_p(n-L+1) + c_F(n) \end{aligned}$$

   for $n \in \{L, L+1, \cdots, N-M\}$ and if we use a vector notation, we get:

   $$\mathbf{x}_v = \mathbf{X}_p \mathbf{h} + \mathbf{c}_F$$

   where $\mathbf{x}_v = [x_v(L) \quad \cdots \quad x_v(N-M)]^T$ and where:

   $$\mathbf{X}_p = \begin{bmatrix} x_p(L+M) & \cdots & x_p(1) \\ \vdots & \ddots & \vdots \\ x_p(N) & \cdots & x_p(N-L-M+1) \end{bmatrix}$$

   is a Toeplitz matrix constructed from the observations $x_p(n)$.

2. We infer that $\mathbf{h} = \mathbf{X}_p^{\#}\mathbf{x}_v$. $\mathbf{X}_p^{\#}$ refers to the pseudo-inverse of $\mathbf{X}_p$ which can be obtained, either by using the `pinv(Xp)` function, or by typing `h=Xp \ Xv`. Once $\mathbf{h}$ has been estimated, we can find an estimation of $\mathbf{c}_F$ using the expression $\mathbf{c}_F = \mathbf{x}_v - \mathbf{X}_p\mathbf{X}_p^{\#}\mathbf{x}_v$.

   The processing is performed by the program:

```
%===== SEPARECG.M
% The file FOETUS.DAT contains:
%     xp(2500 * 1) (chest sensor),
%     almost equal to the mother's EKG,
%     xv(2500 * 1) (abdominal sensor)
%     equal to the fetus's EKG plus
%     the filterd mother's EKG: Xv = h * Xp + cf
%     Sampling frequency = 300 Hz
load fetus.dat
xp=fetus(:,1)-mean(fetus(:,1));
xv=fetus(:,2)-mean(fetus(:,2));
N=length(xv);
%===== Estimation of h
L=20;          % L causal
M=3;           % M anticausal
Xv=xv(L:N-M);
col=xp(L+M:N); lig=xp(M+L:-1:1);
Xp=toeplitz(col,lig);
h=Xp \ Xv;     % Resolution
cf=Xv-Xp*h;    % Fetal heart beats
%===== Displaying the results
Nmax=1000; indx=[1:Nmax];
subplot(311); plot(xp(indx)); grid
subplot(312); plot(xv(indx)); grid
subplot(313); plot([zeros(L-1,1);cf(L:Nmax)]); grid
```

We represented in Figure 12.27, in the bottom graph, the signal extracted after processing. As you can see, the signal originating from the mother has been correctly extracted from the signal picked up by the abdomen sensor (middle graph). However, you can make out in some places the presence of a very faint residue of the mother's heart beats in the bottom graph. ■

12.9.3 Estimating cardiac rhythms

We now have to estimate the fundamental frequency of the EKG signal assumed to be periodic. We have already encountered this problem with pitch detection in speech, and we used a correlation measurement to perform this estimation. Here, we are going to give a theoretical justification for a least squares approach.

Theoretically, if a centered signal $s(n)$ is periodic with period P, the func-

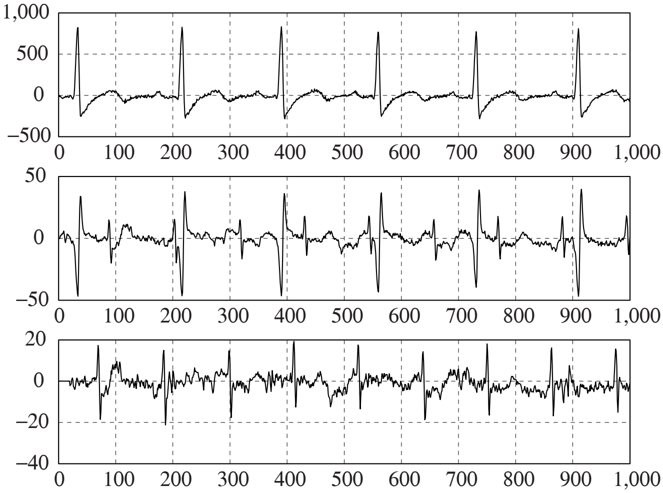


Figure 12.27 – *EKG signals: below, the EKG signal of the fetus, extracted from the abdominal signal (middle graph) using the mother's EKG (top graph)*

tion defined by:

$$g(k) = \lim_{K \to +\infty} \frac{\sum_{n=-K}^{+K} s(n+k)s(n)}{\sqrt{\sum_{n=-K}^{+K} s^2(n)}\sqrt{\sum_{n=-K}^{+K} s^2(n+k)}} \tag{12.20}$$

is itself periodic with period P. It reaches its maximum in $k = 0$, and reaches it again for the various multiples of P.

Example 12.5 (Measuring the fundamental frequency)
Consider a signal $s(n)$, with $n = 1 \dots N$, periodic with period P, and let us construct the two length L vectors $\mathbf{v}_0 = [s(0) \quad \dots \quad s(L-1)]^T$ and $\mathbf{v}_k = [s(k) \quad \dots \quad s(L+k-1)]^T$. If k is a multiple of the period P, we will have:

$$\frac{\mathbf{v}_k}{||\mathbf{v}_k||} = r(k)\frac{\mathbf{v}_0}{||\mathbf{v}_0||} + \varepsilon$$

where the coefficient $r(k)$, which is equal to $+1$ in the ideal case, is used in the sequence to account for the slight variability of the signal's amplitude. If we assume that $\mathbf{w}_0 = \mathbf{v}_0/||\mathbf{v}_0||$ and $\mathbf{w}_k = \mathbf{v}_k/||\mathbf{v}_k||$, the previous equation used as a model for the periodicity of the signal $s(n)$ can be written:

$$\mathbf{w}_k = r(k)\mathbf{w}_0 + \varepsilon \tag{12.21}$$

Notice that $||\mathbf{w}_0|| = ||\mathbf{w}_k|| = 1$.

1. Using the least squares method, determine the estimator $\hat{r}(k)$ of $r(k)$ that minimizes the norm of the error ε.

2. Show that $|r(k)| < 1$. What does the limit case where $r(k) \lesssim +1$ signify? Use this interpretation to determine a method for estimating the period P of the signal $s(n)$. How must the length L of the vector $\mathbf{v}_0$ be chosen?

3. If the rhythm that has to be estimated is between 80 and 200 beats per minute, and if the sampling period is equal to 1/300 s, determine the range of values that must be chosen for P.

4. The previous estimate leads to a value of the period, expressed in seconds, which is a multiple of the sampling period. How can this accuracy be improved?

5. Write a MATLAB® function that estimates, using the previous method, a signal's fundamental frequency.

HINT:

1. Directly applying the least squares formula to equation 12.21 leads to the following estimation of $r(k)$:

$$\begin{aligned}\hat{r}(k) &= \frac{1}{\mathbf{w}_0^T \mathbf{w}_0} \mathbf{w}_0^T \mathbf{w}_k = \frac{\mathbf{v}_0^T \mathbf{v}_k}{\|\mathbf{v}_0\| \|\mathbf{v}_k\|} \\ &= \frac{\sum_{n=1}^{L} s(n+k)s(n)}{\sqrt{\sum_{n=1}^{L} s^2(n)}\sqrt{\sum_{n=1}^{L} s^2(n+k)}}\end{aligned}$$

   This expression should be compared with the expression 12.20 defining $g(k)$. We can infer a method for estimating P:

   (a) Calculate the sequence $\hat{r}(k)$ by varying k over a range of values between P_m and P_M.

   (b) The maximum of the resulting sequence is determined. If this maximum is close enough to 1, the signal is said to be periodic, the maximum's argument is chosen as the period.

   The choice of L is essentially related with the practical duration N for which it is reasonable to assume that the signal is periodic *in a stationary sense*. Obviously, if we wish to detect the possible fluctuations of the rhythm, the windows have to be chosen smaller. Once N is chosen, the square deviation of the estimation decreases as L increases. Hence, if P_M refers to the maximum value for the period analysis, we can choose $L = N - P_M$.

2. Using the Schwarz inequality, we have to check that $r(k)$ has a modulus smaller than 1. When $r(k) \lesssim +1$, $\mathbf{v}_0 \approx +\mathbf{v}_k$. We can then consider that the signal is periodic and that the period is a sub-multiple of k. In the opposite case, the signal is likely not to be periodic. This result leads to a procedure for detecting and estimating the pitch. $r(k)$ is calculated with k belonging to an exploration range (P_m, P_M) of possible values. If the maximum value of $r(k)$ is greater than a threshold ρ chosen beforehand, the signal is considered periodic. In this case, the first maximum of $r(k)$, greater then ρ, leads to an estimation of the period P. The value of ρ is chosen experimentally based on a large number of observations of the signals to be processed.

3. For a rhythm of B beats per minute, that is $b = B/60$ beats per second, the period P expressed as a number of points is given by the integer part of bF_s, where F_s refers to the sampling frequency. As such, to explore the pulse range (B_m, B_M), we have to explore the range of values of k defined by `(fix(Bm*Fe/60):fix(BM*Fe/60))`.

4. Notice that the values that can be obtained for the period, expressed in seconds, are multiples of $1/F_s$. Therefore, the determination accuracy of the correlation function's maximum depends on the choice of the sampling rate, but because the signal is assumed to have been properly sampled, it can interpolated with a rate R and thus increase the evaluation accuracy of the maximum. Of course, in the same manner as with the frequency resolution, this increases the accuracy with which the x-coordinate of the maximum is obtained, but *in no way does it enhance*, in the presence of noise, the mean square error, which is related to the random position of the maximum.

5. The `f0cor.m` function detects and estimates the fundamental frequency:

```
function [F0,corr]=f0cor(sn,Fe,R,thr_corr,Fmin,Fmax)
%%==========================================================%
%% SYNOPSIS: [F0,corr]=F0COR(sn,Fe,R,thr_corr,Fmin,Fmax)    %
%%  sn       = Signal from which the frequency is extracted %
%%  Fe       = Sampling frequency (Hz)                      %
%%  R        = Oversampling factor                          %
%%  thr_corr = Threshold                                    %
%%  Fmin     = Min. frequeny (Hz)                           %
%%             otherwise Fmin=2*Fe/longueur(sn);            %
%%  Fmax     = Max. frequency (Hz)                          %
%%             otherwise Fmax=Fe/2-Fmin;                    %
%%  F0       = Fundamental frequency (Hz)                   %
%%  corr     = Correlations sequence                        %
%%==========================================================%
```

```
sn=interp(sn,R); Fe=R*Fe;
N=length(sn); sn=sn(:); sn=sn-mean(sn);
lagmin=fix(Fe/Fmax); lagmax=fix(Fe/Fmin);
corr=zeros(1,lagmax-lagmin+1);
%===== The effects of the window's size can be tested
%      by taking wlg<wlgmax=N-lagmax
wlg=N-lagmax; v0=sn(1:wlg);
for ii=lagmin:lagmax
    vP=sn(ii:ii+wlg-1);
    corr(ii-lagmin+1)=(v0'*vP)/sqrt((v0'*v0)*(vP'*vP));
end
[niv1, indmax]=max(corr);
if niv1<thr_corr
   pf0=0; F0=NaN;
   return
else
    for ii=lagmin+1:lagmax
        if corr(ii-lagmin+1)>niv1*0.9
            while corr(ii-lagmin+1)>corr(ii-lagmin)
               ii=ii+1;
            end
            pf0=ii-2; F0=Fe/pf0;
            return
        else
            F0=Fe/(indmax+lagmin-1);
        end
    end
end
return
```

In the method considered for measuring the pitch in this example, the correlation function shows maxima in the multiples of the period we wish to determine. Therefore, when scanning the range of possible values, several of these maxima can be encountered. Because of measurement uncertainty, it is possible that the highest of these maxima does not correspond to the period. Therefore, we must search for the possible maxima at sub-multiples of the one corresponding the highest of these maxima. The `f0cor.m` function we have given performs such an operation by searching for other possible maxima greater than 0.9 times the highest maximum. Other more efficient processes can be designed to solve this problem (see further on in this paragraph).

Figure 12.28 shows the levels of the correlation functions, in their respective exploration ranges, for the mother's EKG (top graph), and for the EKG of the fetus (bottom graph).

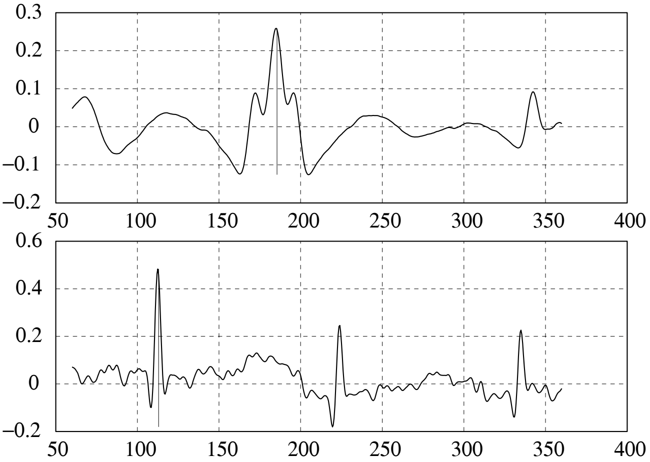


Figure 12.28 – *Levels of the correlation function for the mother's EKG (top graph), and for the EKG of the fetus (bottom graph)*

These graphs were obtained using the following program:

```
%===== EXTRYTHM.M
% Rhythm estimation
% This program uses signals from separeg.m
% with the file fetus.mat
% Uses: f0cor.m
pulsemin=50;     %==== Beats per mn
pulsemax=300;    %==== Beats per mn
%===== Oversampling rati
R=2;
maxcor_apriori=0.25;
[F_mother corr_mother]=...
   f0cor(xp,Fe,R,maxcor_apriori,pulsemin/60,pulsemax/60);
[F_fetus corr_fetus]=...
   f0cor(cf,Fe,R,maxcor_apriori,pulsemin/60,pulsemax/60);
%===== Displaying the results
disp('****************************')
disp(sprintf('*   Pulse of the mother: %5.2f',60*F_mother));
disp(sprintf('*   Pulse of the fetus : %5.2f',60*F_fetus));
disp('****************************')
%===== Displaying the results
lagminM=fix(60*Fe*R/pulsemax);
lagminF=fix(60*Fe*R/pulsemax);
MinCM=min(corr_mother);
LCM=length(corr_mother); MaxCM=max(corr_mother);
subplot(211);
```

```
plot((lagminM:LCM+lagminM-1)/R,corr_mother)
grid; hold on;
plot(Fe/F_mother*[1 1]+1,[min(corr_mother) MaxCM],’:’);
hold off
%=====
MinCF=min(corr_fetus); MaxCF=max(corr_fetus);
subplot(212);
plot((lagminF:length(corr_fetus)+lagminF-1)/R,corr_fetus)
grid; hold on;
plot(Fe/F_fetus*[1 1]+1,[MinCF MaxCF],’:’);
hold off
```

Running the program returns:

```
****************************
*   Pulse of the mother: 97.56
*   Pulse of the fetus :  160.71
****************************
```

■

We saw previously that the presence of several maxima in the correlation we are measuring can pose a problem. One way of eliminating this error consists of calculating the spectral product defined by:

$$P(f) = \prod_{k=1}^{K} \left|X(e^{2j\pi kf})\right|^2 \text{ where } f \leq 1/2K \tag{12.22}$$

where $X(f)$ is the DTFT of a block of the signal. If the latter is periodic with the fundamental frequency $f_0 = F_0/F_s$, its spectrum will show high amplitude peaks in frequencies that are multiples of the frequency f_0. Hence the product $P(f_0/2)$ of the spectrum's values calculated in multiples of $f_0/2$ will have a small value. This is due to the almost null values associated with the frequencies that are odd multiples of $f_0/2$. This is not the case of the product $P(f_0)$ calculated in f_0, which accumulates the spectrum's amplitudes in f_0, $2f_0$, etc. Therefore, the function $P(f)$ allows us to do away with the ambiguity in $f_0/2$: if the frequency estimate $\hat{f}$ is close to $f_0/2$, the value of $P(\hat{f})$ will be much smaller than $P(2\hat{f})$. Comparing these two values then allows us to choose $2\hat{f}$ as the fundamental frequency. However, in order for this method to work well, it requires precise calculation of the spectrum, especially if the spectrum has very sharp peaks, which implies long calculation times.

Comment: when the signal we wish to estimate the pitch of contains much higher frequencies than the pitch, we can also, as it is done with some speech encoders, perform a low-pass filtering before estimating the pitch. For a narrow band signal around a central frequency F_c, that is a signal in the form of a brief oscillating impulse, we can also replace the signal `x` with its *envelope*.

The `sanal.m` function given in example 1.1 calculates the analytical signal associated with a real signal. The envelope can then be acquired by calculating its modulus.

12.10 Extracting the contour of a coin

We are going to try to determine the ellipse representing the contour of a coin (Figure 12.29). The process can be achieved in two very different ways: the first one performs an approximate extraction of the contour, then applies the least squares method. The second performs an approximate extraction of the elliptical disk representing the coin, then extracts the contour using a correlation method.

Example 12.6 (Extracting the elliptical disk) Based on the image of a coin that you will save in "levels of gray" , write a program that approximately extracts the elliptical disk corresponding to the inside of the coin, as it is shown in Figure 12.29.

HINT: type:

```
%===== PREPROCESSCOIN.M
clear all; close all;
load mcoin2; % or imread...
figure(1); set(gcf,'color',[1 1 1])
subplot(221); imagesc(pixc); Spix=size(pixc);
colormap('gray'); axis('image'); title('Original')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
%===== Threshold
minpix=100; ipx0=find(pixc < minpix); yim0=ones(Spix)*255;
yim0(ipx0)=zeros(size(ipx0));
subplot(222); imagesc(yim0); colormap('gray');
axis('image'); title('Seuillage')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
%===== Gaussian filter
hg=moygauss(5); yim0g=filter2(hg,yim0);
subplot(223); imagesc(yim0g); title('Filtre gaussien')
colormap('gray'); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
%===== Threshold
minpix2=170;ipx02=find(yim0g < minpix2); yim02=ones(Spix)*255;
yim02(ipx02)=zeros(size(ipx02));
subplot(224); imagesc(yim02); colormap('gray');
axis('image'); title('Seuillage')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
```

■

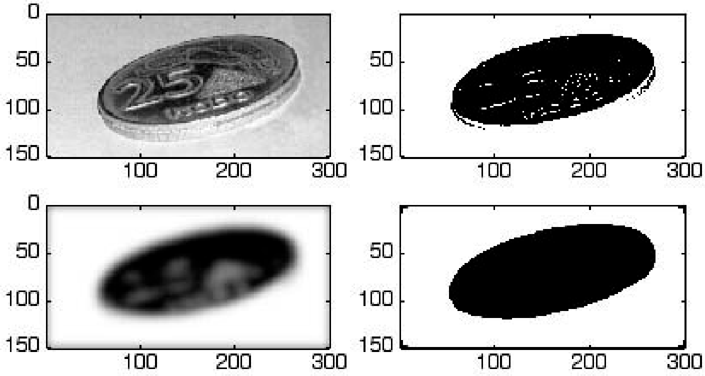


Figure 12.29 – *Result of pre-processing*

Exercise 12.14 (Ellipse contour: the least squares method)
Remember that an ellipse can be described by the equation:

$$ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 - 1 = 0 \tag{12.23}$$

where (x_1, x_2) represents the point in the plane with the x-coordinate x_1 and the y-coordinate x_2. A first process leads to the least squares estimation over N points of the ellipse, to the coefficients a, b, c, d and e of equation 12.23.

1. Write a program, based on the program in example 12.6, which extracts an approximate contour of the coin. You can use the "numerical differentiation" function (`diff`) or the Gaussian differentiation function (`dergauss`).

2. Using the "least squares method", determine the coefficients a, b, c, d, and e of the ellipse closest to the contour that we found.

3. Write a function that plots the ellipse defined by equation 12.23. Do so by rewriting the equation of the ellipse in the form $(\mathbf{x} - \mathbf{x}_0)^T\mathbf{E}(\mathbf{x} - \mathbf{x}_0) = \gamma$, where $\mathbf{x} = [x_1 \quad x_2]^T$, then determine the expressions of $\mathbf{E}$, $\mathbf{x}_0$ and γ as functions of a, b, c, d, and e. Finally, plot the ellipse with the use of the function `ellipse.m` designed in example 21, Chapter "Introduction to MATLAB".

Exercise 12.15 (Ellipse contour: the covariance method)
Consider a sequence of N points on the plane, described by N random variables $\mathbf{x}_1, \ldots, \mathbf{x}_N$ assumed to be independent and uniformly distributed on the elliptical disk defined by its contour:

$$(\mathbf{x} - \boldsymbol{\mu})^T\mathbf{M}^{-1}(\mathbf{x} - \boldsymbol{\mu}) = 1$$

where $\mathbf{M}$ refers to a positive matrix and $\boldsymbol{\mu}$ is the center of the ellipse. Let:

$$\mathbf{y}_n = \mathbf{M}^{-1/2}(\mathbf{x}_n - \boldsymbol{\mu}) \tag{12.24}$$

You can easily check that the sequence $\mathbf{y}_n$ constitutes a sequence of independent random variables uniformly distributed on the circular disk with a unit radius.

1. Determine the expression of the vector $\mathbb{E}\{\mathbf{y}_1\}$ and of the matrix $\mathbb{E}\{\mathbf{y}_1\mathbf{y}_1^T\}$.

2. According the law of large numbers, if $\mathbf{y}_n$ refers to a sequence of independent random vectors with the mean $\boldsymbol{\nu}$ and the covariance $\mathbf{C}$, then when N tends to infinity:

$$\frac{1}{N}\sum_{n=1}^{N}(\mathbf{y}_n - \hat{\boldsymbol{\nu}}_N)(\mathbf{y}_n - \hat{\boldsymbol{\nu}}_N)^T \xrightarrow{p.s.} \mathbf{C}$$

   where $\hat{\boldsymbol{\nu}}_N = N^{-1}\sum_{n=1}^{N}\mathbf{y}_n$. By applying the law of large numbers to the sequence $\mathbf{y}_1, \dots, \mathbf{y}_N$, infer that:

$$\mathbf{M}^{-1} \approx \frac{4}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_N)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_N)^T \text{ where } \hat{\boldsymbol{\mu}}_N = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$$

3. Apply these results to the problem of determining the ellipse closest to the contour of the coin.

12.11 Principal component analysis (PCA)

The *principal component analysis* (*PCA*) provides a simple way of reducing a complex set of data by projecting it onto a space with a small dimension while preserving as much of the variability as possible. This method has the advantage of being linear, and makes no hypothesis concerning the data distribution. It is widely used in a number of applications.

12.11.1 Determining the principal components

Consider N length d vectors $\mathbf{x}_1, \dots, \mathbf{x}_N$ each one of them associated with a set of d measurements, and $\mathbf{X}$ the matrix:

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\ \vdots & \ddots & & \vdots \\ x_{d,1} & x_{1,2} & \cdots & x_{d,N} \end{bmatrix} = [\mathbf{x}_1 \cdots \mathbf{x}_N]$$

$\mathbf{X}$ can be understood from two points of view:

1. either as a set of N columns $\mathbf{x}_n$, each one of them representing *d factors* associated with a same individual. This leads to N points in the space $\mathbb{R}^d$;
2. or as d lines, each one of them representing the same factor for N individuals. This leads to d points in the space $\mathbb{R}^N$.

We wish to reduce the number of factors to keep only the most significant $k < d$ ones.

First consider the case where $k = 1$. We have to search in $\mathbb{R}^d$ the direction of the unit vector $\mathbf{v}$ such that the projection of the set of the $\mathbf{x}_n$ onto this direction leads to the scatter of N points with the highest dispersion (Figure 12.30 for $d = 2$).

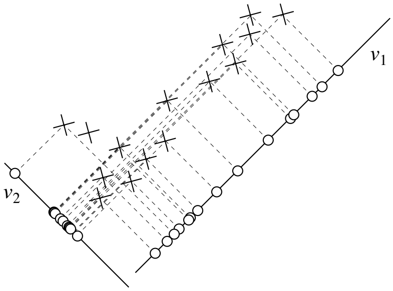


Figure 12.30 – *Projecting the samples for the directions* $\mathbf{v}_1$ *and* $\mathbf{v}_2$*: the dispersion of the projected points is more favorable to an analysis for the vector* $\mathbf{v}_1$ *than it is for* $\mathbf{v}_2$

This can be interpreted as follows: if we can only keep one component, it might as well be the one that best separates all of the points. A classic criterion to evaluate this dispersion is to consider the sum of the distances between all of the projected points. Remember that the projection of $\mathbf{x}_n$ is given by $\mathbf{v}\mathbf{v}^T\mathbf{x}_n$ and that the distance between any two points is written $\|\mathbf{v}\mathbf{v}^T\mathbf{x}_i - \mathbf{v}\mathbf{v}^T\mathbf{x}_j\|^2 = \mathbf{v}^T(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T\mathbf{v}$. The criterion to be minimized is then written:

$$J(\mathbf{v}) = \frac{1}{2N^2}\mathbf{v}^T\left(\sum_{i=1}^{N}\sum_{j=1}^{N}(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T\right)\mathbf{v}$$

Let $\bar{\mathbf{x}} = N^{-1}\sum_{j=1}^{N}\mathbf{x}_j$. Therefore:

$$\begin{aligned} J(\mathbf{v}) &= \frac{1}{2N^2}\mathbf{v}^T\left(\sum_{i=1}^{N}\sum_{j=1}^{N}[(\mathbf{x}_i - \bar{\mathbf{x}}) - (\mathbf{x}_j - \bar{\mathbf{x}})][(\mathbf{x}_i - \bar{\mathbf{x}}) - (\mathbf{x}_j - \bar{\mathbf{x}})]^T\right)\mathbf{v} \\ &= \mathbf{v}^T\mathbf{R}_N\mathbf{v} \end{aligned}$$

where we have let:

$$\mathbf{R}_N = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T \tag{12.25}$$

Notice that the $(d \times d)$ matrix $\mathbf{R}_N$ can be interpreted as a covariance matrix. Hence the problem can be laid out as follows:

$$\begin{cases} \max_{\mathbf{v}} J(\mathbf{v}) \\ \text{with} \quad \mathbf{v}^T\mathbf{v} - 1 = 0 \end{cases}$$

The Lagrange multipliers lead us to the following equivalent problem:

$$\begin{cases} \max_{\mathbf{v}} \left(\mathbf{v}^T\mathbf{R}_N\mathbf{v} - \lambda(\mathbf{v}^T\mathbf{v} - 1)\right) \\ \mathbf{v}^T\mathbf{v} - 1 = 0 \end{cases}$$

which leads, by setting to zero the gradient with respect to $\mathbf{v}$:

$$\begin{cases} \mathbf{R}_N\mathbf{v} - \lambda\mathbf{v} = \mathbf{0} \\ \mathbf{v}^T\mathbf{v} - 1 = 0 \end{cases}$$

The first equation means that $\mathbf{v}$ is an eigenvector of the matrix $\mathbf{R}_N$. If the eigenvalue associated with $\mathbf{v}$ is denoted by λ, then $J(\mathbf{v}) = \lambda \geq 0$. As such, the maximum is reached when $\mathbf{v}$ is chosen as the eigenvector of $\mathbf{R}_N$ associated with the highest eigenvalue: hence the name of this method, *principal component analysis* (*PCA*).

By calculating the product $\mathbf{v}^T\mathbf{X}$, we get a line vector of $\mathbb{R}^N$. This line vector can be interpreted as the "mean" factor that best characterizes, by itself, the N individuals (it results in a good separation). However, this "factor" does not really exist; it is merely a linear combination of factors that are actually observed.

The previous result can easily be generalized: the k principal directions are the k eigendirections of the highest eigenvalues.

Implementing the previous calculations can pose a problem in the case of image processing. Let us assume for example that we are dealing with images comprised of 100×100 pixels represented in the form of length 10^4 vectors. The matrix $\mathbf{R}_N$ associated with these vectors is a $(10^4 \times 10^4)$ matrix! The following property can therefore be useful when $d \gg N$.

Property 12.1 *Let* $\mathbf{A}$ *be a* $(d \times N)$ *matrix. Let* $\{\lambda_1, \cdots, \lambda_N\}$ *be the* N *eigenvalues of the* $(N \times N)$ *matrix* $\mathbf{A}^H\mathbf{A}$. *Then the* d *eigenvalues of the* $(d \times d)$ *matrix* $\mathbf{A}\mathbf{A}^H$ *are:*

$$\{\lambda_1, \cdots, \lambda_N, \underbrace{0, \cdots, 0}_{d-N}\}$$

HINT: let λ be an eigenvalue of $\mathbf{A}^H\mathbf{A}$ associated with the eigenvector $\mathbf{v}$, which is written $\mathbf{A}^H\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$. If we multiply both sides of this equality on the left by $\mathbf{A}$, we get:

$$\mathbf{A}\mathbf{A}^H\mathbf{A}\mathbf{v} = \lambda\mathbf{A}\mathbf{v}$$

Let $\mathbf{w} = \mathbf{A}\mathbf{v}$. We get:

$$\mathbf{A}\mathbf{A}^H\mathbf{w} = \lambda\mathbf{w}$$

Therefore, λ is the eigenvalue of $\mathbf{A}\mathbf{A}^H$ associated with $\mathbf{w} = \mathbf{A}\mathbf{v}$. ■

Example 12.7 (Eigenfaces) Construct a database containing front views of the faces of different people[4]. We can use the ORL database, available to anyone on AT&T's website. This database contains photographs showing the faces of 40 people. Each one of them was photographed 10 times. These photos are stored as images in levels of grey with 112 × 92 pixels. In our example, we constructed a catalog called orlfaces, comprised of the catalogs named s1, s2, ... s40, each one of them containing the 10 photographs we are going to process.

Write a MATLAB® program:

- that changes each $(d_1 = 112) \times (d_2 = 92)$ photograph into a vector;
- that constructs, using a photograph of each of the N people, a subspace $\mathcal{H}$ the dimension of which is less than or equal to N, and such as to have the maximum dispersion of the N projections (think of using the property 12.1);
- that checks the identity corresponding to a photograph by determining its projection onto $\mathcal{H}$ then by comparing the distances of this projection with respect to the N projections obtained with the N previous photographs;
- which constructs the confusion, for which the element (i, j) represents the number of times the person i was chosen as being the person j.

HINT: type:

```
%===== EXEIGENFACES.M
clear all
d1=112; d2=92; d=d1*d2; figure(1); colormap('gray')
imagesNb=10; peopleNb=10; images=cell(peopleNb,imagesNb);
matX=zeros(d,peopleNb,imagesNb); eigenfaces1=zeros(d,peopleNb);
for ni=1:peopleNb
    for kimg=1:imagesNb
```

[4] When using PCA for face recognition, it is important for the photos corresponding to the same person to be taken in approximately the same position and the same lighting. Otherwise, an alignment and a calibration are often unavoidable to achieve satisfactory results.

```
        filename=sprintf('orlfaces/s%i/%i.png',ni,kimg);
        images{ni,kimg}=imread(filename); aux=images{ni,kimg};
        matX(:,ni,kimg)=reshape(aux,d,1);
    end
end
%===== Training of the eigenfaces using the first image
%      of each person
moymatX=matX(:,:,1)*ones(peopleNb,1)/peopleNb;
nbeig=6; eigenfaces=zeros(d,nbeig); matXc=zeros(d,peopleNb);
for ni=1:peopleNb, matXc(:,ni)=matX(:,ni,1)-moymatX; end
RR=matXc'*matXc; [UU, DD, VV]=svd(RR); DD=diag(DD);
UUpbT=matXc(:,:,1)*VV(:,1:nbeig);
UUpbT=UUpbT*diag(1./sqrt(DD(1:nbeig)));
for ni=1:peopleNb,
    cni(ni,:)=matXc(:,ni,1)'*UUpbT;
    vi=reshape(UUpbT*cni(ni,:)',d1,d2);
    subplot(211);imagesc(reshape(matX(:,ni),d1,d2));
    axis('image'); subplot(212);imagesc(vi); axis('image'); pause
end
%===== Testing the pictures of the person
matriceconf=zeros(peopleNb,peopleNb);
for testedNb=1:peopleNb
    for ii=2:imagesNb
        aux1=(matX(:,testedNb,ii)-moymatX)'*UUpbT;
        for ni=1:peopleNb
            aux2(ni)=norm(aux1-cni(ni,:));
        end
        [aa zz]=min(aux2);
        matriceconf(zz,testedNb)=matriceconf(zz,testedNb)+1;
    end
end
matriceconf
```

■

12.11.2 2-Dimension PCA

Consider N images of the same size $d_1 \times d_2$. Let $\mathbf{A}_\ell$ be the matrix representing the image ℓ and:

$$\mathbf{A}_{c,\ell} = \mathbf{A}_\ell - N^{-1} \sum_{j=1}^{N} \mathbf{A}_i$$

be the *centered* image. We need to determine k_1 length d_1 unit vectors $\mathbf{v}$ such that the vectors $\mathbf{y}_\ell = \mathbf{v}^T \mathbf{A}_{c,\ell}$ are maximally dispersed. This is done by maximizing the quantity:

$$\sum_{\ell=1}^{N} \mathbf{v}^T \mathbf{A}_{c,\ell} \mathbf{A}_{c,\ell}^T \mathbf{v}$$

A calculation similar to the one done in the previous paragraph, for length 1 vectors, proves that the vectors we are trying to determine are the k_1 eigenvectors of the $(d_1 \times d_1)$ square matrix $\mathbf{R}_1$:

$$\mathbf{R}_1 = N^{-1} \sum_{\ell=1}^{N} \mathbf{A}_{c,\ell} \mathbf{A}_{c,\ell}^T$$

Let $\mathbf{V}$ be the $(d_1 \times k_1)$ matrix obtained by compiling these k_1 vectors according to the expression:

$$\mathbf{V} = [\mathbf{v}_1 \quad \cdots \quad \mathbf{v}_k] \tag{12.26}$$

Likewise, if we wish to determine the k_2 length d_2 unit vectors such that the vectors $\mathbf{y}_\ell = \mathbf{A}_{c,\ell}\mathbf{w}$ are maximally dispersed, we end up finding that these vectors are the k_2 eigenvectors of the $(d_2 \times d_2)$ square matrix $\mathbf{R}_2$:

$$\mathbf{R}_2 = N^{-1} \sum_{\ell=1}^{N} \mathbf{A}_{c,\ell}^T \mathbf{A}_{c,\ell}$$

Let $\mathbf{W}$ be the $(d_2 \times k_2)$ matrix obtained by compiling these k_2 vectors according to the expression:

$$\mathbf{W} = [\mathbf{w}_1 \quad \cdots \quad \mathbf{w}_k] \tag{12.27}$$

In practice, k_1 and k_2 are chosen such that $k_1 < d_1$ and $k_2 < d_2$, leading to a sequence of N "reduced" images of the size $k_1 \times k_2$, according to the expression:

$$\mathbf{B}_\ell = \mathbf{V}^T \mathbf{A}_\ell \mathbf{W} \tag{12.28}$$

Example 12.8 (2D-PCA) Write a program that extracts from a set of N $(d_1 \times d_2)$ images the matrices $\mathbf{V}$ and $\mathbf{W}$ that allow us, according to expression 12.28, to obtain N $(k_1 \times k_2)$ images. Have the program display the N images in the form of an array of cells, each cell representing a $(d_1 \times d_2)$ image, and return the matrices $\mathbf{V}$ and $\mathbf{W}$.

HINT: type the following function:

```
function [V,W]=PCA2D(matXcell,k1,k2)
%%=========================================%
%% SYNOPSIS: [V,W]=PCA2D(matXcell,k1,k2)   %
%%   matXcell = dimension N cell array     %
%%   a cell is a dimension (d1 x d2) array %
%%       k1 = Reduced nomber of rows       %
%%       k2 = Reduced nomber of columns    %
%%        V = dimension (k1 x d1) array    %
%%        W = dimension (k2 x d2) array    %
```

```
%%=======================================%
N=length(matXcell); gXcellc=cell(1,N); [d1,d2]=size(matXcell{1});
moy_image=zeros(d1,d2); GG=zeros(d1,d2);
for ii=1:N, moy_image=moy_image+double(matXcell{ii}); end
moy_image=moy_image/N;
%===== Centered images
for ii=1:N, gXcellc{ii}=double(matXcell{ii})-moy_image; end
gR=zeros(d1,d1);
for ii=1:N, gR=gR+gXcellc{ii}*gXcellc{ii}'; end
%===== A covariance
gR=gR/N; [gU, lambda, gV]=svd(gR);
ap=lambda(1:k1,1:k1); V=gU(:,1:k1);
%===== The other covariance
gR=zeros(d2,d2);
for ii=1:N, gR=gR+gXcellc{ii}'*gXcellc{ii}; end
gR=gR/N; [gU, lambda, gV]=svd(gR);
W=gU(:,1:k2);
```

This function is used in exercise 12.16. ■

When trying to recognize someone among g individuals, the PCA approach makes you determine *separately*, during the training phase, the principal directions for each group of individuals. The approach we are now going to see makes it possible to simultaneously optimize the choice of the principal directions and the separation into groups.

12.11.3 Linear discriminant analysis (LDA)

We now consider g groups of individuals, each group comprised of N_ℓ individuals for which d factors were measured. The data can then be represented in the form of g matrices of the type:

$$\mathbf{X}_\ell = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N_\ell} \\ \vdots & \ddots & & \vdots \\ x_{d,1} & x_{1,2} & \cdots & x_{d,N_\ell} \end{bmatrix} = [\mathbf{x}_{\ell,1} \cdots \mathbf{x}_{\ell,N_\ell}]$$

In the space $\mathbb{R}^d$ we obtain g scatters containing respectively $N_1, \ldots, N_g$ points. We assume that $N = \sum_{\ell=1}^{g} N_\ell$.

The goal of the *linear discriminant analysis* (*LDA*) is to find the best separation for these g scatters of points. To achieve this, we must first introduce the following definitions:

– the mean, or barycenter, of a group:

$$\mathbf{m}_\ell = N_\ell^{-1} \sum_{j=1}^{N_\ell} \mathbf{x}_{\ell,j}$$

– the overall mean of the g groups:

$$\mathbf{m} = N^{-1}\sum_{\ell=1}^{g}\sum_{j=1}^{N_\ell}\mathbf{x}_{\ell,j} = \frac{\sum_{\ell=1}^{g} N_\ell \mathbf{m}_\ell}{\sum_{\ell=1}^{g} N_\ell}$$

– the intraclass covariance (internal to the considered class) defined by:

$$\mathbf{R}_I = \frac{\sum_{\ell=1}^{g} N_\ell \mathbf{R}_\ell}{\sum_{\ell=1}^{g} N_\ell} \text{ with } \mathbf{R}_\ell = N_\ell^{-1}\sum_{j=1}^{N_\ell}(\mathbf{x}_{\ell,j}-\mathbf{m}_\ell)(\mathbf{x}_{\ell,j}-\mathbf{m}_\ell)^T$$

which leads us to:

$$\mathbf{R}_I = N^{-1}\sum_{\ell=1}^{g}\sum_{j=1}^{N_\ell}\mathbf{x}_{\ell,j}\mathbf{x}_{\ell,j}^T - N^{-1}\sum_{\ell=1}^{g} N_\ell \mathbf{m}_\ell \mathbf{m}_\ell^T$$

– the extraclass covariance defined by:

$$\mathbf{R}_E = N^{-1}\sum_{\ell=1}^{g} N_\ell(\mathbf{m}_\ell-\mathbf{m})(\mathbf{m}_\ell-\mathbf{m})^T$$

which can be interpreted as the dispersion of the barycenters of each class with respect to the overall mean;

– the total covariance defined by:

$$\mathbf{R} = N^{-1}\sum_{\ell=1}^{g}\sum_{j=1}^{N_\ell}(\mathbf{x}_{\ell,j}-\mathbf{m})(\mathbf{x}_{\ell,j}-\mathbf{m})^T$$

A simple calculation shows that:

$$\mathbf{R} = N^{-1}\sum_{\ell=1}^{g}\sum_{j=1}^{N_\ell}\mathbf{x}_{\ell,j}\mathbf{x}_{\ell,j}^T - \mathbf{m}\mathbf{m}^T$$

These covariance matrices are $(d \times d)$ matrices. It can easily be proved that:

$$\mathbf{R} = \mathbf{R}_I + \mathbf{R}_E$$

We are now going to find the direction, parallel to the vector $\mathbf{v}$, such that the intraclass dispersion is minimal and the interclass dispersion is maximal: graphically speaking, the scatters are farther away from each other, and more compact. To achieve this objective, one possible criterion is to minimize the evaluation function defined by:

$$J(\mathbf{v}) = \frac{\mathbf{v}^T\mathbf{R}_I\mathbf{v}}{\mathbf{v}^T\mathbf{R}\mathbf{v}}$$

This amounts to searching $\mathbf{v}$ such that:

$$\min_{\mathbf{v}} \mathbf{v}^T\mathbf{R}_I\mathbf{v} \text{ with } \mathbf{v}^T\mathbf{R}\mathbf{v} - 1 = 0 \tag{12.29}$$

Using the Lagrange multiplier method, we end up with the following equivalent problem:

$$\begin{cases} \min_{\mathbf{v}} \left(\mathbf{v}^T\mathbf{R}_I\mathbf{v} - \lambda(\mathbf{v}^T\mathbf{R}\mathbf{v} - 1)\right) \\ \text{with} \quad \mathbf{v}^T\mathbf{R}\mathbf{v} - 1 = 0 \end{cases}$$

By setting the gradient with respect to $\mathbf{v}$ to zero, we find that $\mathbf{v}$ is such that:

$$\mathbf{R}^{-1}\mathbf{R}_I\mathbf{v} = \lambda\mathbf{v}$$

Notice that $\mathbf{R}^{-1}\mathbf{R}_I$ is a positive matrix. If we now express the constraint $\mathbf{v}^T\mathbf{R}\mathbf{v} - 1 = 0$, we infer that $J(\mathbf{v}) = \lambda \geq 0$. Hence we have to chose the eigenvector associated with the smallest eigenvalue. An exactly identical calculation can be done to prove that $\mathbf{v}$ is also an eigenvector of $\mathbf{R}^{-1}\mathbf{R}_E$ associated with the highest eigenvalue.

Let $\mathbf{v}_1, \ldots, \mathbf{v}_k$ be the k eigenvectors associated with the k highest eigenvalues of $\mathbf{R}^{-1}\mathbf{R}_E$. By compiling these k vectors, we get the $(d \times k)$ matrix:

$$\mathbf{V} = \begin{bmatrix}\mathbf{v}_1 & \cdots & \mathbf{v}_k\end{bmatrix} \tag{12.30}$$

This means that for each of the g families, the vectors:

$$\mathbf{y}_{\ell,j} = \mathbf{V}^T\mathbf{x}_{\ell,j} \tag{12.31}$$

which give the representative points of each image. Theoretically, each of the g scatter has a minimal dispersion, and all of the scatters are as far away from each other as possible.

Example 12.9 (LDA) Write a function that performs the linear discriminant analysis of g groups, each group containing N_ℓ vectors of the same length d. We will be using g cells as the data format with MATLAB®, where each cell is a $(d \times N_\ell)$ matrix. The program returns the matrix $\mathbf{V}$ defined by equation 12.30 which is used for changing the g scatters in $\mathbb{R}^d$ into g "well" separated scatters in $\mathbb{R}^k$.

HINT: type the following function:

```
function V=LDA(gX,kk)
%%============================================================%
%% SYNOPSIS: V=LDA(gX,kk)                                      %
%%   gX = array of cells. A cell is a (d x Nell) matrix        %
%%         associated with les Nell vectors of a class         %
%%   kk = reduced dimension (kk<d)                             %
%%   V  = (kk x d) matrix used to reduce X with Xreduced=V'X %
%%============================================================%
d=size(gX{1},1); G=length(gX);
XredLDA=cell(1,G); moyell=cell(1,G);
Nell=cell(1,G); gXc=cell(1,G);
moyT=zeros(d,1); NT=0; RI=zeros(d,d);
for ell=1:G
    gXell=gX{ell}; Nell{ell}=size(gXell,2);
    NT=NT+Nell{ell}; moyell{ell}=mean(gXell')';
    moyT=moyT+Nell{ell}*moyell{ell};
    gXellc=gXell-double(moyell{ell})*ones(1,Nell{ell});
    gXc{ell}=gXellc; RI=RI+gXellc*gXellc';
end
RI=RI/NT; moyT=moyT/NT; RE=zeros(d,d);
for ell=1:G
    vaux=moyell{ell}-moyT; RE=RE+Nell{ell}*vaux*vaux';
end
RE=RE/NT; RR=RI+RE; AA=inv(RR)*RE;
[gV, lambda,bid]=svd(AA);
lambda=diag(lambda); ap=lambda(1:kk);
V=gV(:,1:kk);
```

■

Exercise 12.16 (Face recognition) Consider a database comprised of g groups of photographs in levels of grey of the same person's face, taken from the front. We can once again use the ORL database, available on AT&T's website. In this exercise, the database of $N = 10$ photos is divided in two: A photographs are used for training, and the $N - A$ others are used for recognitions tests.

Training: write a program:

- that determines, using the function in example 12.8, the matrices $\mathbf{V}$ and $\mathbf{W}$ associated with each person (try for instance $k_1 = 4$ and $k_2 = 3$);
- that determines the $k = k_1 k_2$ vector characterizing each person and each photograph;
- that learns from the g previous groups of vectors the matrices of the linear discriminant analysis. Use the function from example 12.9. By choosing the dimension $d = 2$, you can display representative

barycenters in the plane, but in order to obtain good recognition results, you will need a higher value for d, for example $d = 6$;

- that finds the barycenters characterizing each individual.

Recognition: write a program that successively determines from expressions 12.28 and 12.31 the position of the test images in the space $\mathbb{R}^d$ ($d = 2$). Note that, usually, the person's identity is not known. Therefore, the obtained position has to be compared with the scatters characterizing each person. As a test function, you can use, for example, the distance between obtained position and the barycenter of the scatter, and construct the confusion matrix.

The programs developed in exercise 12.16 can also be used for character recognition. The program `geneBDDchif.m` allows you to generate a database of the 10 digits in printing characters, each digit being recorded in 12 different copies. After having created the folders `s0`, ..., `s9` in the folder `mdigits`, type:

```
%===== DIGITBDDGENE.M
clear all; close all
name={'times','courrier','verdana'}; sizech=[120 100 60];
angle={'normal','italic'}; bolfCh={'normal','bold'};
for ii=0:9
   for jj=1:3
      figure(1); tt=sprintf('%i',ii);
      set(gcf,'color',[1 1 1],'position',[60 500 180 180])
      plot(0,0); set(gca,'unit','pixel')
      set(gca,'xtick',[],'xticklabel',[])
      set(gca,'ytick',[],'yticklabel',[])
      set(gca,'box','off','xcolor',[1 1 1],'ycolor',[1 1 1])
      ff=text(-1,0,tt);
      set(ff,'fontsize',sizech(jj),'fontname',name{jj})
      for kk=1:2
         set(ff,'fontangle',angle{kk})
         for mm=1:2
            set(ff,'fontweight',bolfCh{mm});
            num=2*2*(jj-1)+2*(kk-1)+mm;
 %cde=sprintf('print -dtiffnocompression mdigits/s%i/%i.tif',ii,num);
 %cde=sprintf('print -dpng mdigits/s%i/%i.png',ii,num);
            % Windows only:
      cde=sprintf('print -dbitmap mdigits/s%i/%i.bmp',ii,num);
                eval(cde)
            end
        end
    end
end
```

Use the program `LDAPCAtest.m` while adapting the values of the parameters. Try it in particular with `nbimages_A=4;`, `dim_barycenter=30;` and `k1=6; k2=6;` (set in `LDAPCAtraining.m`). The `-dbitmap` parameter is only supported by the `MS-Windows` operating system. The called functions will have to be modified accordingly. It is usually preferable to use the options `dtiffnocompression` or `dpng`.

12.12 Separating an instantaneous mixture

Consider a sequence of length d vector observations $\mathbf{x}_n$ obtained from a mixture of signals $\mathbf{s}_n$ of the same size according to the expression:

$$\mathbf{x}_n = \mathbf{A}\mathbf{s}_n$$

where $\mathbf{A}$ is a $(d \times d)$ matrix assumed to be invertible. We are going to try to determine the sequence $\mathbf{s}_n$ without knowing $\mathbf{A}$. This is called a *blind processing*. Of course, the problem is badly laid out, since there are more unknowns than there are equations. However, if we make certain hypotheses concerning the $\mathbf{s}_n$, we can estimate the matrix $\mathbf{B} = \mathbf{A}^{-1}$.

Let us first assume that $\mathbf{s}_n$ is stationary (we can assume without being any less general that $\mathbf{s}_n$ is centered). We get $\mathbb{E}\left\{\mathbf{x}_n\mathbf{x}_n^H\right\} = \mathbf{A}\mathbb{E}\left\{\mathbf{s}_n\mathbf{s}_n^H\right\}\mathbf{A}^H$ and therefore, with obvious notations:

$$\mathbf{R}_x = \mathbf{A}\mathbf{R}_s\mathbf{A}^H \tag{12.32}$$

Now let us assume that $\mathbf{A}$ is a solution of 12.32. Then for any unitary matrix $\mathbf{U}$ ($\mathbf{U}\mathbf{U}^H = \mathbf{I}$), we can easily check that the matrix $\tilde{\mathbf{A}} = \mathbf{A}\mathbf{R}_s^{1/2}\mathbf{U}\mathbf{R}_s^{-1/2}$ is still a solution of 12.32. As a consequence, knowing the second order moments only allows us to know the matrix $\mathbf{A}$ multiplied by an unknown unitary matrix. Therefore, it is not possible with a blind process to find $\mathbf{s}_n$ using only second order moments. Some authors have considered using moments with orders higher than 2. We will not be doing so in this work[5].

Let us remove the independence hypothesis and consider that we have L length N sequences $\mathbf{x}_{n,\ell}$. The blocks of length N can be obtained by dividing up the observations $\mathbf{x}_n$ according to expression:

$$\mathbf{x}_{n,\ell} = \mathbf{x}_{\ell N+n}$$

with $\ell \in \{0, \cdots, L-1\}$ and $n \in \{1, \cdots, N\}$. Let us assume that:

$$\mathbf{x}_{n,\ell} = \mathbf{A}\mathbf{s}_{n,\ell}$$

and that each block represents the samples of a white random process, meaning that:

$$\mathbb{E}\left\{\mathbf{s}_{n,\ell}\mathbf{s}_{n,\ell}^H\right\} = \text{diag}\{\sigma_{1,\ell}^2, \cdots, \sigma_{d,\ell}^2\} = \mathbf{D}_\ell$$

[5] Notice that in the case of Gaussian signals, blind separation is never possible, since any moment with an order higher than 2 is expressed with second order moments.

Therefore, for any ℓ:

$$\mathbf{R}_\ell = \mathbb{E}\left\{\mathbf{x}_{n,\ell}\mathbf{x}_{n,\ell}^H\right\} = \mathbf{A}\mathbf{D}_\ell\mathbf{A}^H \;\Leftrightarrow\; \mathbf{B}\mathbf{R}_\ell\mathbf{B}^H = \mathbf{D}_\ell$$

Based on the blocks $\{\mathbf{x}_{1,\ell}, \cdots, \mathbf{x}_{N,\ell}\}$, we can estimate the L matrices $\hat{\mathbf{R}}_\ell$:

$$\hat{\mathbf{R}}_\ell = N^{-1}\sum_{n=1}^{N}(\mathbf{x}_{n,\ell} - \bar{\mathbf{x}}_\ell)(\mathbf{x}_{n,\ell} - \bar{\mathbf{x}}_\ell)^H$$

where $\bar{\mathbf{x}}_\ell = N^{-1}\sum_{k=1}^{N}\mathbf{x}_{n,\ell}$. We can then search for the matrix $\mathbf{B}$ such that, for any ℓ, the positive matrix:

$$\mathbf{C}_\ell = \mathbf{B}\hat{\mathbf{R}}_\ell\mathbf{B}^H$$

is as close as possible to a diagonal matrix. A reasonable criteria would consist of minimizing the sum of the non-diagonal terms with respect to the diagonal terms according to the expression:

$$J(\mathbf{B}) = \sum_{\ell=1}^{L}\sum_{i\neq j}\frac{|\mathbf{C}_\ell(i,j)|^2}{\mathbf{C}_\ell(i,i)\mathbf{C}_\ell(j,j)} \tag{12.33}$$

However, this criterion is very difficult to minimize. To simplify this search, we are going to restrict ourselves to the case where $d = 2$ in the following exercise.

Exercise 12.17 (Separating two sources) Consider the mixture of two speech signals $\mathbf{s}_n^T = [s_n^1 \quad s_n^2]$ obtained with the mixture matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 1.3 \\ -0.1 & 0.8 \end{bmatrix}$$

Based on $\mathbf{x}_n^T = [x_n^1 \quad x_n^2] = \mathbf{A}\mathbf{s}_n$, we wish to restore the signals s_n^1 and s_n^2 with the amplitudes multiplied by an unknown coefficient. This means that we are trying to find the matrix $\mathbf{B} = \mathrm{diag}(\mu_1, \mu_2)\mathbf{A}^{-1}$ where μ_1 and μ_2 are two arbitrary values. Hence all we need to do is find $\mathbf{B}$ in the form:

$$\mathbf{B} = \begin{bmatrix} 1 & \alpha \\ \beta & 1 \end{bmatrix}$$

1. Determine as a function of α, β and the partial covariances of the blocks, the expression of the criterion $J(\alpha, \beta)$ defined by equation 12.33.

2. Write a MATLAB® program that calculates $J(\alpha, \beta)$ over ranges of values for α and β chosen beforehand, and that reconstructs the signals s_n^1 and s_n^2 from the minimum. Choose the length N of a block equal to the stationarity time.

12.13 Matched filters in radar telemetry

What is called an *active radar* emits a signal $s(t)$ in the direction of the area we wish to explore. If a target is there, the signal is sent back in the form of an echo $e(t)$. The presence or the absence of this echo in the received signal indicates the presence or the absence of a possible target. Therefore, we have to choose between the two following hypotheses:

- hypothesis H_0: $x(t) = b(t)$ (noise alone);
- hypothesis H_1: $x(t) = e(t) + b(t)$ (signal+noise).

We will assume that $b(t)$ is a Gaussian, white noise, with the power σ^2, and there is at most one target, that it is not moving (null speed), and that the transmission channel does not distort the signal $s(t)$.

In these conditions, the echo $e(t)$ is equal to $s(t - \tau)$, where τ refers to the delay due to the time it takes the signal to cover the distance to and from the target. In radar observation, this delay is called the two-way echo delay. We can infer the distance d to the target using the formula $d = c\tau/2$, where c refers to the propagation speed of an electromagnetic field ($c \approx 3 \times 10^8$ m/s). If the target's speed v is different from zero, we can show that the signal $s(t)$ is modulated (multiplied) by a sine, the frequency f_d of which is proportional to v, and is called the *Doppler frequency*. From now on, we will assume that $v = 0$.

We saw in exercise 12.12 on matched filtering that the optimal processing, for the given structure (filter + threshold detector), consists of performing the following operations:

1. Filter the received signal $x(t)$ by a filter with the impulse response $s(-t)$ (matched filtering) (Figure 12.31).

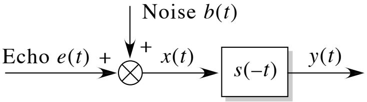


Figure 12.31 – *The principle of RADAR signal processing*

The output signal then has the expression:

$$y(t) = \int_{-\infty}^{+\infty} x(u)s(u-t)du$$

2. Find the maximum y_M of $y(t)$ in an observation window $T_{\max}$ corresponding to the radar's maximum range. For instance, if the this range is equal to 1.5 km, $T_{\max} = 10$ μs.

3. Compare y_M to a threshold. If it is above the threshold, it is decided that the target is present, and its distance is $d = c\tau/2$, where τ refers to the time corresponding to the maximum y_M.

Exercise 12.18 (Radar telemetry)
We are going to study the performances related to the shape of the emitted signal $s(t)$. To do so, we will be considering two continuous-time signals $s_1(t)$ and $s_2(t)$ with a duration of $T = 630$ ns, both shown in Figure 12.32.

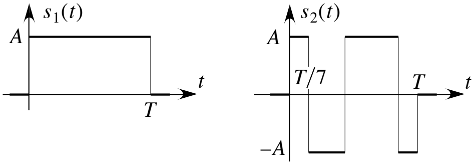


Figure 12.32 – *Two impulse shapes*

Let us assume that the total observation time of the received signal $x(t)$ is $T_{\text{max}} = 10$ μs, and that the sampling frequency is equal to 100 MHz, in all 1,000 samples.

1. Determine the energy E_s of the two impulse $s_1(t)$ and $s_2(t)$. In the presence of noise, we can define the signal-to-noise ratio as $S/B = (E_s/T)/\sigma_b^2$.

2. Determine, in the absence of noise, the maximum output amplitude A_M of the matched filter for each of the two impulses.

3. Write a program:

   – that generates a Boolean random variable indicating the presence or the absence of the target. The impulse detection performances are in no way modified by imposing a value for the delay. You can choose for example $\tau = 2$ μs;

   – that generates the signal $x(t)$ for the types of impulse (take $A = 1$);

   – that determines the matched filter's output signal $y(t)$;

   – that decides whether the impulse is absent or present by comparing the maximum of $y(t)$ to a threshold, and, in case the impulse is decided as present, estimates the value of the delay. For the threshold, you can choose the value $E_s/2$.

4. Interpret the results.

12.14 Kalman filtering

Kalman thought up an approach, introduced in paragraph 11.6 that makes it possible to forgo the stationarity hypothesis made for the Wiener filtering. It is based on a *state representation* (see also paragraph 5.1).

The *Kalman filter* provides the best *mean square* estimation of $\mathbf{x}_n$ from the observation of $\mathbf{y}_n$. In its simplest form, the following hypotheses are made: the components of $\mathbf{b}_n$ and $\mathbf{u}_n$ are *independent, centered, Gaussian*, but not necessarily stationary, random vectors. Hence the respective covariance matrices are as follows:

$$\begin{array}{lll} & \mathbb{E}\left\{\mathbf{b}_n\mathbf{b}_n^T\right\} = \mathbf{R}_b(n) & (m \times m) \\ & \mathbb{E}\left\{\mathbf{b}_n\mathbf{b}_{n+k}^T\right\} = \mathbf{0} \text{ for } k \neq 0 & (m \times m) \\ & \mathbb{E}\left\{\mathbf{u}_n\mathbf{u}_n^T\right\} = \mathbf{R}_u(n) & (p \times p) \\ & \mathbb{E}\left\{\mathbf{u}_n\mathbf{u}_{n+k}^T\right\} = \mathbf{0} \text{ for } k \neq 0 & (p \times p) \\ \text{and} & \mathbb{E}\left\{\mathbf{u}_{n+k}\mathbf{b}_n^T\right\} = \mathbf{0} \text{ for any pair } (n,k) & (p \times m) \end{array}$$

In the particular case where the processes $\mathbf{b}_n$ and $\mathbf{u}_n$ are assumed to be *stationary*, the matrices $\mathbf{R}_b(n)$ and $\mathbf{R}_u(n)$ are independant of n, and we then have, for any n, $\mathbf{R}_b(n) = \mathbf{R}_b(0)$ and $\mathbf{R}_u(n) = \mathbf{R}_u(0)$. If we also assume that their components are not correlated with each other, then the covariance matrices are diagonal and have the respective expressions:

$$\mathbf{R}_b(0) = \begin{bmatrix} \sigma_{b1}^2 & 0 & \cdots & 0 \\ 0 & \sigma_{b2}^2 & 0 & \ddots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_{bm}^2 \end{bmatrix} \text{ and } \mathbf{R}_u(0) = \begin{bmatrix} \sigma_{u1}^2 & 0 & \cdots & 0 \\ 0 & \sigma_{u2}^2 & 0 & \ddots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_{up}^2 \end{bmatrix}$$

The Kalman filter provides at every time n the estimation $\hat{\mathbf{x}}_n$, which is a *linear* function of $\mathbf{y}_1$, $\ldots$, $\mathbf{y}_n$ that minimizes the mean square error $\mathbb{E}\left\{|\mathbf{x}_n - \hat{\mathbf{x}}_n|^2\right\}$. We also know that in the case of a Gaussian signal, this linear estimation is the best one among all the functions $\mathbf{y}_1$, $\ldots$, $\mathbf{y}_n$. However, "bluntly" solving this problem leads to an expression involving the inverse of the $(n \times n)$ covariance matrix. Hence, when n increases, the size of this matrix grows "uncontrollably".

The Kalman filter is a recursive filter. It performs a computation of $\hat{\mathbf{x}}_n$ at the time n by updating the value $\hat{\mathbf{x}}_{n-1}$ obtained at the time $(n-1)$ taking into account the last observed value $\mathbf{y}_n$. And this calculation only requires the memorization of the finite dimension state. Remember that it is expressed as follows, as we have shown in paragraph 11.6 for $n \geq 1$:

Initial values:
$\mathbf{x}_{0|0} = \mathbb{E}\{\mathbf{x}_0\}$ *and* $\mathbf{K}_0 = \mathbb{E}\left\{\mathbf{x}_0\mathbf{x}_0^T\right\}$

Repeat:

$$\begin{aligned}
\mathbf{G}_n &= \mathbf{K}_{n-1}\mathbf{C}_n^T\left(\mathbf{C}_n\mathbf{K}_{n-1}\mathbf{C}_n^T + \mathbf{R}_u(n)\right)^{-1} \\
\mathbf{x}_{n|n} &= \mathbf{A}_{n-1}\mathbf{x}_{n-1|n-1} + \mathbf{G}_n\left(\mathbf{y}_n - \mathbf{C}_n\mathbf{A}_n\mathbf{x}_{n-1|n-1}\right) \\
\mathbf{K}_n &= \mathbf{A}_n\left(\mathbf{I} - \mathbf{G}_n\mathbf{C}_n\right)\mathbf{K}_{n-1}\left(\mathbf{I} - \mathbf{G}_n\mathbf{C}_n\right)^T\mathbf{A}_n^T \cdots \\
&\qquad + \mathbf{A}_n\mathbf{G}_n\mathbf{R}_u(n)\mathbf{G}_n^T\mathbf{A}_n^T + \mathbf{R}_b(n)
\end{aligned}$$

The expression of this algorithm calls for a few comments:

1. Its form is similar to that of the adaptive algorithms (such as the LMS) seen in paragraph 11.2.4.

2. In the particular case where the noises $\mathbf{u}_n$ and $\mathbf{b}_n$ are stationary, the matrices $\mathbf{R}_u(n)$ and $\mathbf{R}_b(n)$ are independent of n.

3. In the particular case where the "system" is time-invariant, the matrices $\mathbf{A}_n$ and $\mathbf{C}_n$ are independent of n.

4. $\mathbf{G}_n$ is called the *Kalman gain*. It can be calculated *beforehand*, since it is determined by equations 12.34 and 12.34, which do not depend on the observed data $\mathbf{y}_n$. However, except in the case of scalars (see exercise 12.19), the expression of $\mathbf{G}_n$ requires us to solve a complex recursive equation.

5. The significance of equation 12.34 is obvious. According to the state equation, if there were no noise, the value of $\mathbf{x}_n$ at the time n would simply be $\mathbf{A}_n\mathbf{x}_{n-1}$. This is the first term of 12.34. But because of the noise, this values has to be corrected by a quantity proportional to the difference between the observed value $\mathbf{y}_n$ and the value we should have obtained, had the observation equation not been noised, that is $\mathbf{C}_n\mathbf{x}_n = \mathbf{C}_n\mathbf{A}_n\mathbf{x}_{n-1}$.

6. We wish to draw your attention to the fact that the algorithm requires us to know $\mathbf{R}_b(n)$ and $\mathbf{R}_u(n)$. However, the theory can be generalized to include the situation where these quantities have to be estimated based on the observed data. In that case, the sequence of gains can no longer be calculated beforehand.

As a conclusion, the main properties of the *Kalman filter* can be summarized as follows:

- it leads to the minimum mean square error;
- it requires a detailed description of the signal's model, without, however, imposing stationarity hypotheses. If the model is changed, the results can become quite wrong;
- the obtained algorithm is *recursive*. This implies that it can be implemented with a small amount of memory space;
- its calculation time is long because of the number of operations to perform, which has sometimes limited its use.

Example 12.10 (Estimating the speed of moving object)
The position of an object moving on the plane is observed. The x and y-coordinates of the trajectory, which are functions of continuous time, are given by $x_{1a}(t)$ and $x_{2a}(t)$ respectively, and the velocity vectors by $v_1(t) = dx_{1a}(t)/dt$ and $v_2(t) = dx_{2a}(t)/dt$ respectively. The samples of $x_{1a}(t)$ and $x_{2a}(t)$ are denoted by $x_1(n)$ and $x_2(n)$ respectively. We will assume that the sampling satisfies the Nyquist condition.

Noised samples of the position are observed, that is:

$$\begin{cases} y_1(n) = x_1(n) + u_1(n) \\ y_2(n) = x_2(n) + u_2(n) \end{cases}$$

where $\mathbf{u}(n) = [u_1(n) \quad u_2(n)]^T$ is a centered, white noise, with the known covariance matrix $\sigma_u^2 \mathbf{I}_2$. From now on we will use the notation $\mathbf{y}(n) = [y_1(n) \quad y_2(n)]^T$.

We are going to try, based on N observations $\{\mathbf{y}(1), \ldots, \mathbf{y}(N)\}$, to estimate the velocity vector as a function of the time n, using the Kalman algorithm.

1. The state of the observed system is denoted by $\mathbf{x}(n) = [x_1(n) \quad x_2(n) \quad v_1(n) \quad v_2(n)]^T$. Find the equation for the evolution of $\mathbf{x}(n)$, if we assume that the object is moving at a constant speed, which is not the case(!). We will assume that any noise in the model only affects the components of the velocity, that it is stationary, and its components are uncorrelated with each other and with the same variance σ_b^2.

2. Find the expression of the measurement as a function of the state. We will assume that the measurement noise is stationary and that its components are uncorrelated with each other and with the same variance σ_u^2.

3. Write a program that calculates the position, adds a noise, and estimates the speed using the Kalman algorithm. Test the algorithm on the following equation:

$$\begin{cases} x_1(n) = n^2 \\ x_2(n) = \sin(2\pi f_0 n) \end{cases}$$

with $f_0 = 0.005$. The observation noise is such that $\sigma_u^2 = 0.5$. Try different values of the modeling noise's variance σ_b^2. Compare the result with the theoretical speed given by:

$$\begin{cases} v_1(n) = 2n \\ v_2(n) = 2\pi f_0 \cos(2\pi f_0 n) \end{cases}$$

HINT:

1. If we assume beforehand that the object is moving at a constant speed, and that the measurement is accurate for the position, but not for the velocity, then the state equation is given by:

$$\begin{cases} x_1(n+1) = x_1(n) + v_1(n) + 0 \\ x_2(n+1) = x_2(n) + v_2(n) + 0 \\ v_1(n+1) = v_1(n) + b_3(n) \\ v_2(n+1) = v_2(n) + b_4(n) \end{cases}$$

that can be written:

$$\mathbf{x}(n+1) = \mathbf{A}\mathbf{x}(n) + \mathbf{b}(n)$$

$$\text{where } \mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and where $\mathbf{b}(n)$ is a noise that represents the uncertainty regarding the hypothesis "the object is moving at a constant speed". We have assumed that:

$$\mathbb{E}\{\mathbf{b}(n)\mathbf{b}^T(n)\} = \sigma_b^2 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. We have:

$$\mathbf{y}(n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \mathbf{x}(n) + \mathbf{u}(n) = \mathbf{C}\mathbf{x}(n) + \mathbf{u}(n)$$

   We have assumed that $\mathbb{E}\{\mathbf{u}(n)\mathbf{u}^T(n)\} = \sigma_u^2 \mathbf{I}_2$.

3. Type:

```
%===== KALTRAJECT2D.M
clear all, clf
N=600; f0=0.005; tps=(1:N);
%===== Position
x1=tps.^2; x2=sin(2*pi*f0*tps);
%===== Theoretical speed
v1=2*tps; v2=2*pi*f0*cos(2*pi*f0*tps);
%===== Noisy observation
var_obs=0.5; Ru=var_obs*diag([1 1]);
Y=[x1;x2]+sqrt(var_obs)*randn(2,N);
AA=[1 0 1 0; 0 1 0 1; 0 0 1 0; 0 0 0 1];
CC=[1 0 0 0; 0 1 0 0];
%===== Choice for the variance of the model noise
var_mod=0.000001;
Rb=var_mod*diag([0 0 1 1]);
%===== States
xchap=zeros(4,N);
%===== Initialization
Xnm1=zeros(4,1); % xchap(n-1)
%===== Without an a priori on E(x(0)x'(0)), I(n,n) is taken
KK=eye(4);
%===== Kalman algorithm
for k=2:N,
    GG=KK*CC'*inv(CC*KK*CC'+Ru);
    xchap(:,k)=AA*xchap(:,k-1)+GG*(Y(:,k)- ...
           CC*AA*xchap(:,k-1));
    KK=AA*(eye(4)-GG*CC)*KK*(eye(4)-GG*CC)'*AA'...
       +AA*GG*Ru*GG'*AA'+Rb;
end;
%=====
figure(2); subplot(1,2,1); plot(x1,x2,':'); hold on
plot(Y(1,:),Y(2,:),'r'); grid; hold off
subplot(1,2,2);  plot(v1,v2,':'); hold on;
plot(xchap(3,40:N),xchap(4,40:N),'r');
plot(xchap(3,N),xchap(4,N),'xr'); grid; hold off
```

   The graph on the right of Figure 12.33 leads to the estimation of the object's velocity without noise (dashed line) and the estimated one (full line). Its noisy trajectory is shown on the left.

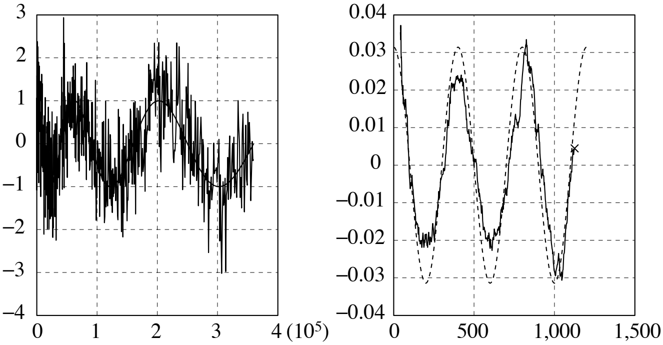


Figure 12.33 – *Kalman algorithm in trajectography*

■

Exercise 12.19 (Denoising of an AR-1 signal using Kalman)

The discrete-time signal $y(n) = x(n) + u(n)$ is observed, where $n \in \mathbb{Z}$. The observation $y(n)$ is obtained with the following program:

```
%===== MOBILEX.M
% Generation of the observed signal
T=200;
a=0.9;                  % Time constant of the model
sigmab=1;               % Model noise, variance=1
b=sigmab*randn(T,1);
x=filter(1,[1 -a],b);   % Trajectory
sigmau=3;               % Observation noise, variance=9
y=x+sigmau*randn(T,1);  % Observation
```

This program corresponds to the system described by:

$$\begin{cases} x(n) = ax(n-1) + b(n) & \text{(state equation)} \\ y(n) = x(n) + u(n) & \text{(observation equation)} \end{cases}$$

$b(n)$ and $u(n)$ are assumed to be two white noises, uncorrelated with each other. Hence, $x(n)$ is an AR-1 process. Let $\rho = \sigma_b^2/\sigma_u^2$. We want estimate $x(n)$ using the observed $y(n)$.

1. Determine, as a function of a and σ_b^2, the expression of the power $P_0 = \mathbb{E}\left\{x^2(0)\right\}$ corresponding to the stationary solution of the state equation. This value will serve as the initial expression for $\mathbb{E}\left\{x^2(0)\right\}$.

2. Determine, as a function of a and ρ, the recursive equation of the Kalman gain, as well as the initial value $G(0)$.

3. Write a program that implements the Kalman filter on the suggested trajectory.

12.15 Compression

12.15.1 Scalar quantization

The scalar quantization of a random quantity X consists of partitioning $\mathbb{R}$ in n sub-intervals:

$$I_1 = (a_0 = -\infty, a_1), \ldots, I_k = (a_{k-1}, a_k), \ldots, I_n = (a_{n-1}, a_n = +\infty)$$

and of defining a sequence $\{\mu_1, \ldots, \mu_n\}$ of real values. From then on:

- the *coding* process associates x, the value assumed by the random variable X, with the index k of the interval to which the word belongs, that is, associates k with x if $x \in I_k$;
- the *decoding* process associates the index k with the value μ_k.

The value μ_k can be called either the *code word* or the *representative element* of the interval I_k, or simply the *reconstruction value.* The set of code words is called a *codebook.*

Hence you can see that the coding consists of associating the value x with the adress of one of the words in the codebook. In practice, the codebook size n is often chosen equal to a power of 2, that is $n = 2^N$, and the memory addresses are then written using N bits.

The linear quantization we studied in paragraph 7.5 is the simplest case of scalar quantization. All of the n intervals, except for the first one and the last one, are chosen with the same length q, and the representative elements are taken from the middle of the interval. Coding amounts to testing whether x belongs to one of the n intervals of the type $(kq-q/2, kq+q/2)$ and to transmit k.

Mathematically, these two operations, coding and decoding, are summarized by the application:

$$X \to \mu(X)$$

What this actually means is that no difference is made between all the elements of the interval I_k and the code word μ_k. In this context, coding improves as the distorsion caused by these operations grows fainter, for a given number n of intervals. Of course, for the elements close to μ_k, the error is small. On the other hand, for the elements far away, the error is high. Hence it is best to have intervals of small length for the most frequent values of x and to let the intervals be longer for the less probable values. We already mentioned this when studying linear quantization. We are now going to give a mathematical expression for this relation.

The problem is to find, based on a probability distribution of X known beforehand, the best partitioning and the best representative elements with respect to a given criterion of distorsion between X and $\mu(X)$. In the search for a solution, the pioneers are undoubtedly Lloyd and Max, and the reader can look up the famous reference [59, 64].

We will give here the solution in the case where the criterion that has to be minimized is the square deviation defined by $\mathbb{E}\{(X-\mu(X)\}^2)$, and the expression of which is:

$$J(\{a_k\},\{\mu_k\}) = \mathbb{E}\{(X-\mu(X))^2\} = \sum_{k=1}^{n}\int_{a_{k-1}}^{a_k}(x-\mu_k)^2 p_X(x)dx \quad (12.34)$$

where $p_X(x)$ refers to the probability density of X. The problem consists of determining the $(2n-1)$ values $a_1, \dots, a_{n-1}, \mu_1, \dots, \mu_n$ that minimize $J(\{a_k\},\{\mu_k\})$. This solution can be obtained by setting to zero the partial derivatives of J with respect to the a_k and the μ_k. This leads to a system of equations which, once simplified, can be written:

$$\left\{\begin{array}{ll} a_k = \dfrac{\mu_k+\mu_{k+1}}{2} & k=1,\dots,(n-1) \\[2ex] \mu_k = \dfrac{\int_{a_{k-1}}^{a_k} x p_X(x)dx}{\int_{a_{k-1}}^{a_k} p_X(x)dx} & k=1,\dots,n \end{array}\right. \quad (12.35)$$

The first expression simply means that the interval I_k is the set of points closest to μ_k. And according to the second one, μ_k can be interpreted as the *barycenter* of the interval I_k weighted by $p_X(x)$.

Unfortunately, the expressions 12.35 generally do not lead to a simple analytical form for the quantities $a_1, \dots, a_{n-1}$ and $\mu_1, \dots, \mu_n$. Numerical methods can then be used, such as the gradient algorithm, the general form of which is:

$$\theta_p = \theta_{p-1} - \lambda \left.\frac{\partial J}{\partial \theta}\right|_{\theta=\theta_{p-1}}, \qquad \lambda > 0 \quad (12.36)$$

where θ refers to the parameter for which we want to determine the numerical value that minimizes the criterion $J(\theta)$. The index p refers in this case to the p-th iteration. The *positive* number λ is the *gradient step.* We have already encountered and made comments on such an algorithm in paragraph 11.4.2 on page 420.

In our case, the parameter we have to determine is $\theta = (a_1, \dots, a_{n-1}, \mu_1, \dots, \mu_n)^T$, which is comprised of $(2n-1)$ values. Using 12.34, we can infer the $(2n-1)$ component expression of the gradient:

$$\frac{\partial J}{\partial \theta} = \begin{bmatrix} \dfrac{\partial J}{\partial a_1} & \cdots & \dfrac{\partial J}{\partial a_{n-1}} & \dfrac{\partial J}{\partial \mu_1} & \cdots & \dfrac{\partial J}{\partial \mu_n}\end{bmatrix}^T \quad (12.37)$$

with:

$$\left\{\begin{array}{ll} \dfrac{\partial J}{\partial a_k} = ((a_k - \mu_k)^2 - (a_k - \mu_{k+1})^2)p_X(a_k) & k = 1, \ldots, (n-1) \\ \dfrac{\partial J}{\partial \mu_k} = -2\int_{a_{k-1}}^{a_k}(x - \mu_k)p_X(x)dx & k = 1, \ldots, n \end{array}\right. \tag{12.38}$$

If $p_X(x)$ is a Gaussian distribution, the second relation of the system 12.38 can be numerically evaluated using MATLAB® with the `erf` function. The following program implements the gradient algorithm 12.36:

```
%===== LLOYD.M
usrpi=1/sqrt(2*pi); rc2=sqrt(2);
%===== 7 parameters to be calculated
n=4; Ga=zeros(n-1,1); Gmua=zeros(n,1); Gmub=zeros(n,1);
a=[-3; 0; 3];         %==== Initialization
mu=[-4; -2; 2; 4];
lambda=0.1;           %==== Gradient step
for jj=1:2000
    mu1=mu(1:n-1); mu2=mu(2:n);
    expa=usrpi*exp(-(a .^2)/2);
    Ga=((a-mu1) .^2 - (a-mu2) .^2) .* expa;
    Gmua=-2*([0;expa]-[expa;0]);
    Gmub=mu .* (erf([a/rc2;+inf])-erf([-inf;a/rc2]));
    Gmu=Gmua+Gmub; mu=mu-lambda*Gmu;
    a=a-lambda*Ga;
end
%===== Estimated parameters
[a;mu]'
```

For $n = 4$ and $\sigma = 1$, the program returned:

$$\left\{\begin{array}{l} a_3 = -a_1 = -0.9816 \quad a_2 = 0 \\ \mu_4 = -\mu_1 = 1.5104 \quad \mu_3 = -\mu_2 = 0.4528 \end{array}\right. \tag{12.39}$$

In the case of a centered, Gaussian random process with the variance σ^2, all we have to do is multiply these values by σ. As we have already said, the gradient method converges slowly and the calculation time is long. However, it can be shown that, if the function $\log(p_X(x))$ is strictly concave, the function J has only one minimum.

12.15.2 Vector quantization

Introductory example

The vector quantization problem is formulated in the same way as the scalar quantization. The difference is that we now wish to code length m vectors instead of real scalars. Hence we have to determine the best way of partitioning $\mathbb{R}^m$ in n regions and associate each one of these regions with the best representative element.

The simplest idea consists of coding the m components *separately*, using scalar quantization. Consider, for example, the case where $m = 2$ and where 4 bits are used to code a point of $\mathbb{R}^2$. We can allocate 2 bits to each of the vector's two components and then code each of the two components using 2 bit scalar quantization. But we can also try to directly code the point in $\mathbb{R}^2$ using 4 bits. We are going to see, with an example first, that this second method can be better.

To obtain a sequence of correlated vectors we consider the AR-1 process defined by the equation $s(n)+as(n-1) = w(n)$, where $|a| < 1$ and where $w(n)$ is a centered, Gaussian, random sequence with the variance σ^2. We are going to use the following representation: based on the signal $s(n)$, we construct the two component vector $\mathrm{s}_2(p) = [s(2p-1) \quad s(2p)]^T$ obtained by grouping together the two consecutive samples of $s(n)$. The following program displays a sequence of 1,000 values of $\mathrm{s}_2(p)$ in the form of a scatter in $\mathbb{R}^2$. The results are shown in Figure 12.34. The correlation found in the AR-1 process reveals itself by the elliptical shape of the scatter (see exercise 9.5 on drawing the confidence ellipse):

```
%===== PARTQV2AR1.M
N=1000; a=0.8; N2=N/2;
w=randn(N,1); s=filter(1,[1 a],w);
s2=zeros(2,N/2); s2(:)=s; s2=s2';
plot(s2(:,1),s2(:,2),'x'); axis('square')
%===== Matrice de covariance
%C=s2'*s2/(N/2);       % Estimation de C
C=[1 -a;-a 1]/(1-a*a); % Valeur theorique
%===== Ellipse de confiance
alpha=.9; s=-2*log(1-alpha);
hold on; ellipse([0 0],inv(C),s);
ellipse([0 0],inv(C),s);
hold off
grid
```

Coding the vector $\mathrm{s}_2(p) \in \mathbb{R}^2$ using 4 bits means that we have to partition $\mathbb{R}^2$ into 16 regions. In the case where the two components of $\mathrm{s}_2(p)$ are coded separately using 2 bits, this partition is made of rectangular regions with their sides parallel to the axes. This is shown in Figure 12.34(a). The positions of the boundary lines are determined by the optimal values of the 2 bit scalar quantization of a Gaussian random variable, provided by 12.39.

Figure 12.34(b) also shows a partition based on 16 other regions, which takes more into account the elliptical distribution of the points in the plane. The regions are delimited first by the ellipse's major axis and second by the 8 lines parallel to the minor axis located in scalar positions indicated by the 3 bit Lloyd-Max algorithm.

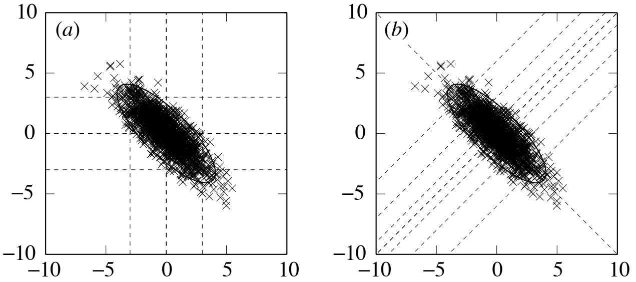


Figure 12.34 – *Two partitions of the plane in* 16 *regions (*4 *bits are used to code each representative element of the corresponding region)*

Another idea consists of whitening the two components of the sequence $\mathbf{s}_2(p)$ so as to transform the elliptical scatter into a circular scatter. This operation is achieved (see example 8.5, page 283) by applying a square root of the inverse of the covariance matrix:

$$\mathbf{R} = \begin{bmatrix} R(0) & R(1) \\ R(1) & R(0) \end{bmatrix} = \frac{\sigma^2}{1-a^2}\begin{bmatrix} 1 & -a \\ -a & 1 \end{bmatrix}$$

to the sequence of vectors $\mathbf{s}_2(p)$. Note that the operation suggested here does not decorrelate the vectors that are transformed, only their two components. In plainer terms, if we use the notation $\mathbf{t}_2(p) = \mathbf{M}\mathbf{s}_2(p)$, where $\mathbf{M}$ is the modal matrix, $\mathbb{E}\left\{\mathbf{t}_2(p)\mathbf{t}_2^T(p)\right\}$ is diagonal but $\mathbb{E}\left\{\mathbf{t}_2(p)\mathbf{t}_2^T(p+k)\right\} \neq \mathbf{0}$.

In practice, the matrix $\mathbf{R}$ can be estimated, from the length N sequence $s(n)$ (and therefore the length of $\mathbf{s}_2(p)$ is equal to $N/2$), as follows:

$$\hat{\mathbf{R}} = \frac{1}{N/2}\sum_{p=1}^{N/2} \mathbf{s}_2(p)\mathbf{s}_2^T(p)$$

We then obtain a sequence of vectors $\hat{\mathbf{R}}^{-1/2}\mathbf{s}_2(p)$ the two components of which are uncorrelated. We can then perform the 2 bit quantization of each of the two components. You can check by typing:

```
%===== PARTQV2W.M
N=5000; a=0.8; N2=N/2;
w=randn(N,1); s=filter(1,[1 a],w);
s2=zeros(2,N/2); s2(:)=s; s2=s2';
subplot(121),plot(s2(:,1),s2(:,2),'x');
grid; axis('square')
%===== Decorrelation
R2est=s2'*s2/N/2; s2t=s2*sqrtm(inv(R2est));
```

```
subplot(122); plot(s2t(:,1),s2t(:,2),'x');
grid; axis('square')
```

As an exercise, you can perform a simulation on 1,000 values and compare the square deviation of these three quantization rules by choosing as representative elements the points located in the center of these regions.

The last method which, after estimating the covariance matrix, whitens the two components can be extended to a higher number of components. This method can also be generalized to any form of transformation that tends to perform a decorrelation of the data. This is the case, for example, of the discrete Fourier transform, of the discrete cosine transform. The coding can then be performed component by component in the transformed region: in this context, this is called transform coding [66].

Voronoi regions and centroids

We now come back to the problem of determining the best partition and the best representative elements in vector quantization. Let:

$$\mathcal{R} = \{R_1, \ldots, R_n\} \tag{12.40}$$

be the partition of the observation space $\mathbb{R}^m$ in n disjoint regions and:

$$\mathcal{C} = \{\mu_1, \ldots, \mu_n\}$$

the codebook of the n representative elements (Figure 12.35) where μ_k is the element in $\mathbb{R}^m$ that replaces every point of the region R_k.

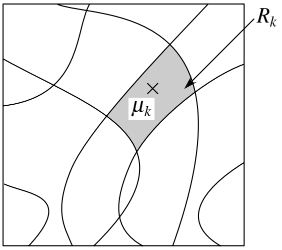


Figure 12.35 – *Partition $\mathcal{R}$ of the plane*

Let us assume that the probability distibution of the vector observation X, with possible values in $\mathbb{R}^m$, has a probability density denoted by $p_X(x)$. Then the mean square deviation has the expression:

$$J(\mathcal{R},\mathcal{C}) = \sum_{k=1}^{n} \int_{R_k} ||x - \mu_k||^2 p_X(x)dx \tag{12.41}$$

where $||v|| = \sqrt{v^T v}$ represents the Euclidean-norm of the vector v. The goal is to minimize $J(\mathcal{R}, \mathcal{C})$ with respect to $\mathcal{R}$ and $\mathcal{C}$. This can be performed numerically, by repeating the two following operations:

1. we start with n representative elements $\mathcal{C}$, and find the n regions that minimize $J(\mathcal{R}, \mathcal{C})$;

2. once the n regions have been determined, we find the n new representatives that minimize $J(\mathcal{R}, \mathcal{C})$.

After each step, $J(\mathcal{R}, \mathcal{C})$ becomes smaller. The two operations are iterated until $J(\mathcal{R}, \mathcal{C})$ reaches a value deemed small enough. It is unfortunately possible with this algorithm to end up at a local minimum. Let us examine these two operations in detail:

1. If $\mathcal{C}$ is set, minimizing $J(\mathcal{R}, \mathcal{C})$ with respect to $\mathcal{R}$ amounts, according to expression 12.41, to assigning to R_k all of the points of $\mathbb{R}^m$ such that:

   $$R_k = \{x \in \mathbb{R}^m \quad : \quad ||x - \mu_k|| < ||x - \mu_j|| \quad \forall j \neq k\} \tag{12.42}$$

   The regions defined by such a partition are called *Voronoi regions*. This concept can of course be extended to other distances, not just the basic Euclidean distance.

   Figure 12.36 shows the Voronoi regions for a set of points of $\mathbb{R}^2$ when the distance used in the Euclidean distance. In this case, we start with the set of representative elements, and the lines delimiting the regions are simply defined by the perpendicular bisectors of the line segments formed by the pairs of these points. The points belonging to the perpendicular bisectors can indifferently be assigned to either one of the adjacent regions.

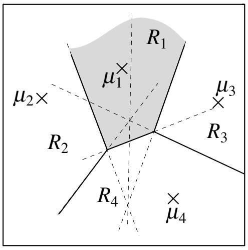


Figure 12.36 – *$\mathcal{R}$ partition by the perpendicular bisectors*

2. If $\mathcal{R}$ is set, and under sufficient regularity conditions, $J(\mathcal{R}, \mathcal{C})$ can be minimized with respect to $\mathcal{C}$ by setting to zero the partial derivative of

$J(\mathcal{R}, \mathcal{C})$ with respect to μ_k. This leads to:

$$\mu_k = \frac{\int_{R_k} x p_X(x) dx}{\int_{R_k} p_X(x) dx} \tag{12.43}$$

The point in $\mathbb{R}^m$ defined by the expression 12.43 is called the *centroid* of the region R_k for the ditribution with the probability distribution $p_X(x)$.

In practice, the probability distribution of the observation X we wish to code is unknown. However, $p_X(x)$ can be estimated based on a set of outcomes. This leads us to the algorithm presented in the following paragraph.

LBG algorithm

The *LBG* algorithm, named after its creators *Linde, Buzzo* and *Gray* (see reference [58]), allows us to determine the n best representative elements of an unknown distribution based on a set of N outcomes called a *training set*:

$$\mathcal{D} = \{x_1, \cdots, x_N\} \quad \text{where} \quad x_i \in \mathbb{R}^m$$

In practice, $N \gg n$. Replacing the unknown probability distribution $p_X(x)$ with the histogram of the values is equivalent to assigning the same probability $1/N$ to every observation. If we then replace this in 12.43, we get:

$$\mu_k = \frac{\sum_{R_k} \frac{1}{N} x_i}{\sum_{R_k} \frac{1}{N}} = \frac{1}{N_k} \sum_{\{i: x_i \in R_k\}} x_i \tag{12.44}$$

where N_k refers to the number of points in the region R_k. Using this result, we can implement the following algorithm. We start with an initial codebook $\mathcal{C}$ of n centroids $\{\mu_1, \ldots, \mu_n\}$ belonging to $\mathbb{R}^m$, then successively perform the following operations:

1. construction of the n Voronoi regions associated with the n centroids, that is to say the partition of the set $\mathcal{D}$ in n subsets containing the points closest to the μ_k;

2. computation, based on expression 12.44, of the n new corresponding centroids;

3. stopping criterion: if the n new centroids are "close" enough to the previous ones, the algorithm stops, otherwise, it picks up at step 1.

It can be shown that the estimated square deviation, defined by:

$$\mathcal{E} = \frac{1}{N} \sum_{k=1}^{n} \sum_{i=1}^{N} \|x_i - \mu_k\|^2 \mathbb{1}(x_i \in R_k) \tag{12.45}$$

decreases after each iteration and that the algorithm converges to a local minimum.

On the other hand, this algorithm poses two problems related to the initialization: first, we are not sure that the algorithm converges to the global minimum, and second, empty spaces can appear while the algorithm is running. To avoid these problems, the LBG algorithm has the following efficient initialization procedure:

1. The centroid is calculated for the set of the whole training sequence.
2. Based on each centroid, two points are constructed by a small shift in two opposite directions. As a consequence, the number of points is doubled.
3. The Voronoi regions, associated with the set of previously obtained points, are determined, as well as their respective centroids.
4. The algorithm goes back to step 2 until a set of n centroids is obtained. The resulting codebook size is a power of 2.

The voronoi.m function

The `voronoi.m` function determines the n Voronoi regions associated with the codebook `C`. It returns the vector `indR` that contains, for each element of the training sequence `x`, the number of the region to which it belongs.

```
function indR=voronoi(x,C)
%%=================================================%
%% SYNOPSIS: indR=VORONOI(x,C)                     %
%%    x    =  (m x T) training sequence            %
%%    C    = n code words (m x n) array            %
%%    indR = Length T vector associating a point x %
%%           to the region it belongs to           %
%%=================================================%
[m, T]=size(x); [m, n]=size(C);
%===== Euclidian norms(^2) of the representatives
norC=ones(1,m)*(C .* C);
%===== Searching the minimal distance for each x
for tt=1:T
    d2=norC - 2*x(:,tt)'*C;
    [bid ind]=min(d2); indR(tt)=ind(1);
end
return
```

The centroides.m function

The `centroides.m` function determines the n centroids (matrix `C`) based on the training sequence along with the number of the region to which each of its elements belong. It returns the value of the square deviation `E` given by expression 12.45:

```
function [C,E]=centroides(x,n,indR)
%%============================================%
%% Computing the n centroids of n regions    %
%% SYNOPSIS : [C,E]=CENTROIDES(x,n,indR)     %
%%    x    = Training sequence               %
%%    n    = Number of centroids             %
%%    indR = Index of the region x belongs to %
%%    C    = Centroids                       %
%%    E    = Square error                    %
%%============================================%
[m,T]=size(x); C=zeros(m,n);
E=0;
for jj=1:n    % For each region
    indRep=find(indR==jj); % Indices of points
                           % belonging to region jj
    xjj=x(:,indRep); [m lxjj]=size(xjj);
if lxjj~=0
       C(:,jj)=(xjj*ones(lxjj,1))/lxjj;   % Estimated mean
       d2=(C(:,jj)*ones(1,lxjj)-xjj) .^2;
       E=E+sum(sum(d2));
    end
end
E=E/T; % Estimated Error
return
```

The initlbg.m function

The **initlbg.m** function determines an initial size n codebook based on the training sequence **x**. The value **delta** sets the shifts in the direction $(1, 1, \cdots, 1)$ and in the direction $(-1, -1, \cdots, -1)$:

```
function C0=initlbg(x,n);
%%===========================================================%
%% Initialization of LBG: determine n centroids by dichotomy %
%% SYNOPSIS: C0=INITLBG(x,n)                                 %
%%     x  = learning sequence (mxT array)                    %
%%     n  = number of representatives (power of 2)           %
%%     C0 = n representatives (mxn array)                    %
%%===========================================================%
delta=0.01; [m,T]=size(x);
C0=x*ones(T,1)/T;    % First representative
indR=ones(1,T);      % Initial array
jj=1; vv=zeros(m,n);
while (jj<n)
   for kk=1:jj
       vv(:,2*kk-1)=C0(:,kk)-delta;
       vv(:,2*kk)=C0(:,kk)+delta;
   end
   vv=vv(:,1:2*jj);
   jj=jj*2; C0=centroides(x,jj,indR);
end
return
```

Implementing the LBG algorithm

The `lbg.m` function implements the LBG algorithm and returns a size n codebook, based on the training sequence `x`. This function also returns the initial codebook produced by the function **`initlbg.m`** as well as the decreasing sequence of values of the square deviation. The length of `E` gives the number of iterations needed to reach the minimum:

```
function [Cf,Ci,E]=lbg(x,n)
%%====================================================%
%% Calculating the dictionary using the LBG algorithm %
%% SYNOPSIS: [Cf,Ci,E]=LBG(x,n)                       %
%%    x  = (mxT) training sequence with:              %
%%        m  dimension of the observations            %
%%        T  nomber of observations                   %
%%    n  = Number of representatives (power of 2)     %
%%    Cf = Dictionary                                 %
%%    Ci = Initial dictionary                         %
%%    E  = Square error based on the iteration step   %
%% Uses: voronoi.m, centroides.m initlbg.m            %
%%====================================================%
[m, T]=size(x); N=log2(n); Cn=zeros(m,n);
indR=zeros(1,T); E(1)=0; ncv=1; epsilon=0.001;
%===== Initialisation
Ci=initlbg(x,n); Cf=Ci;
%===== Iterations
rep=0;
while ncv
    rep=rep+1;
    %===== Searching the Voronoi regions corresponding
    %      to the n centroids. indR gives each point
    %      of x the number the region it belongs to.
    indR = voronoi(x,Cf);
    %===== Calculating the new centroids and the distorsion E
    [Cn,E(rep)]=centroides(x,n,indR);
    if max(abs(Cn-Cf))<epsilon
       ncv=0; Cf=Cn;
    else
       Cf=Cn;
    end
end
return
```

Example 12.11 (Applying the `LBG` function)

We are going to use the `lbg.m` function to find the 4 representative elements of a centered Gaussian distribution with the variance 1. Type:

```
x=randn(1,4000);
[Cf,Ci,E]=lbg(x,4);
Cf
```

Through simulation, we found, values for `Cf` such as:

```
-1.4899    -0.4369     0.4565     1.5036
```

This result is in agreement with the values $\mu_4 = -\mu_1 = 1.5104$ and $\mu_3 = -\mu_2 = 0.4528$ obtained by numerical resolution of the Lloyd-Max equation (see page 526).

The codebook size problem

Let us assume that we have to code the elements of $\mathbb{R}^m$ using 20 bits. A direct application of vector quantization leads to the creation of a $2^{20} \approx 1{,}000{,}000$ word codebook. If this number is too large for the considered application, we can, at the cost of an acceptable performance loss (see exercise 12.20), break the 20 bits down to two or more smaller sub-codebooks. For instance, if the 20 bits are broken down to two sets of 10 bits, two codebooks have to be created, each one containing $2^{10} \approx 1,000$ words. An illustration of this method is given by the vector quantization of the prediction coefficients in a speech coder.

The *MELP coder* (*Mixed Excitation Linear Prediction*) converts speech sampled at 8,000 Hz into a sequence of binary symbols flowing at a rate of 2,400 bits per second. As the name implies, the coder performs a linear prediction analysis on 10 coefficients every 22.5 ms. Therefore, it has at its disposal $2{,}400 \times 22.5\,10^{-3} = 54$ bits to code a frame. 25 of these 54 bits are assigned to coding the 10 prediction coefficients. A direct application of vector quantization would lead to producing a $2^{25} \approx 32{,}000{,}000$ word codebook in $\mathbb{R}^{10}$. Obviously, this is infeasible. Instead, the solution consists of creating several nested codebooks. The principle can be explained, without being any less general, by first considering a vector coding using 6 bits. A direct construction would then consist of using a dictionary of $2^6 = 64$ elements. Instead, and at the cost of an acceptable performance loss, we can construct a first dictionary of 4 representative elements, therefore coded using 2 bits, then code the distance between the vector to be coded and the representative element using the 4 remaining bits with a single dictionary of $2^4 = 16$ representative elements. Thus, the set of two dictionnaries is now only comprised of $16 + 4 = 20$ representative elements instead of 64. To get a better idea, imagine a first partition of the space in 4 regions, then each region is partitioned *in an identical way* in 16 sub-regions. All we need to do when decoding is concatenate the value of the value of the representative element associated with the first two bits and the value of the representative element associated with the last 4 bits. The solution chosen for the MELP coder uses 4 sub-codebooks with the sizes $2^7 = 128$,

$2^6 = 64$, $2^6 = 64$ and $2^6 = 64$, respectively, for a total of only 320 words to code the vector of the 10 prediction coefficients using 25 bits.

Exercise 12.20 (Performances with two sub-codebooks)
Consider the AR-1 process $s(n)$ defined by the recursive equation:

$$s(n) + as(n-1) = w(n)$$

where $a = 0.9$ and where $w(n)$ is a centered, Gaussian random sequence with a variance equal to 1. The "vector" signal $\mathrm{s}_2(p) = [s(2p-1)\ s(2p)]^T$ is reconstructed from the signal $s(n)$ by grouping together two consecutive samples of $s(n)$. We wish to code $\mathrm{s}_2(p)$ using 6 bits.

1. Write a program that generates the length $N = 5{,}000$ sequence $\mathrm{s}_2(p)$ then constructs a codebook of $2^6 = 64$ representative elements with the use of the `lbg.m` function.

2. Using the same sequence, construct two sub-codebooks with $2^2 = 4$ and $2^4 = 16$ representative elements, respectively. Compare the performances. Bear in mind that the computation can take up to a few minutes on a standard computer. However, this usually is not a problem since the computation is done once and for all before the use of the codebooks.

Image applications

To illustrate the performances of the LBG algorithm, we are now going to discuss the example of the compression of an image defined in levels of grey. We are going to start with the size (256×256) image of Lena, and cut it up into T "thumbnails" of m *pixels*. If we want to code each thumbnail using b bits (hence b/m bits per pixel), we have to find 2^b representative elements. The following program applies the `lbg.m` function to the T thumbnails:

```
%===== IMAGETTE.M
clear
b=2;       % Number of bits / representative element
n=2^b;     % Number of  representative elements
ml=2;      % Number of pixels / row
mc=2;      % Number of pixels / column
m=ml*mc;   % Number of pixels
nbbit_pixel=b/m;
%=====
load lena; % Reading the image pixc and colormap cmap
[lig col]=size(pixc); moy_pixc=mean(mean(pixc));
pixc_centre=pixc-moy_pixc;
blocl=lig/ml;         % Number of blocs / row
blocc=col/mc;         % Number of blocks / column
```

```
nbimagette=lig*col/m; % Number of images
%===== The image is divided in (m x 1) vector thumbnails
imaget=zeros(m,nbimagette);
if m==1,
imaget(:)=pixc_centre;
else
   %=====  T column vectors
   for ll=0:blocl-1
       il=ll*ml+1;
       for cc=0:blocc-1
           ic=cc*mc+1; ii=ic+il-1;
           aux=pixc_centre(il:il+ml-1,ic:ic+mc-1);
           imaget(:,ii)=aux(:);
       end
   end
end
%===== Searching the n representative elements
[Cf,Ci,E]=lbg(imaget,n);      %==== LBG
norC=ones(1,m)*( Cf .* Cf );
d2=zeros(n,1); auxcol=zeros(m,1);
%===== Coding the thumbnails
imag0=zeros(ml,mc);            % Initialization
pixc_cod_centre=zeros(lig,col);
for ll=0:blocl-1
    il=ll*ml+1;
    for cc=0:blocc-1
        ic=cc*mc+1; ii=ic+il-1;
        aux=pixc_centre(il:il+ml-1,ic:ic+mc-1);
        auxcol=aux(:);
        for nn=1:n
            d2(nn)=-2*auxcol'*Cf(:,nn) + norC(nn);
        end
        [bid, ind]=min(d2); Ccod=Cf(:,ind); imag0(:)=Ccod;
        %===== Reconstruction
        pixc_cod_centre(il:il+ml-1,ic:ic+mc-1)=imag0;
    end
end
pixc_cod=pixc_cod_centre+moy_pixc;
subplot(121); imagesc(pixc); colormap(cmap);
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0]); axis('square');
subplot(122); imagesc(pixc_cod); colormap(cmap);
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0]); axis('square');
set(gcf,'Color',[1 1 1])
```

If $m = 1$ and $b = 1$, this is equivalent to coding each pixel using 1 bit. The result has to be a "finely detailed" image but with 2 levels of gray. If, on the contrary, $m = 4$ and $b = 4$, hence still 1 bit per pixel, the image definition will not be as good, with $2^4 = 16$ thumbnails to represent the image. Figures 12.37 and 12.38 illustrate the differences obtained depending on the values of m and b.

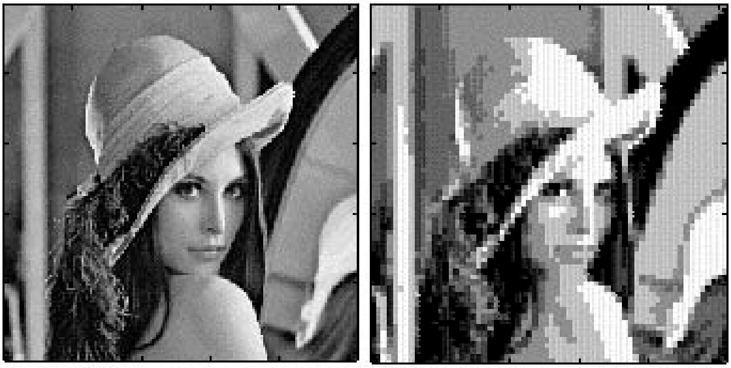


Figure 12.37 – *Original image on the left and the result of the LBG compression with* $m = 16$ *and* $b = 3$

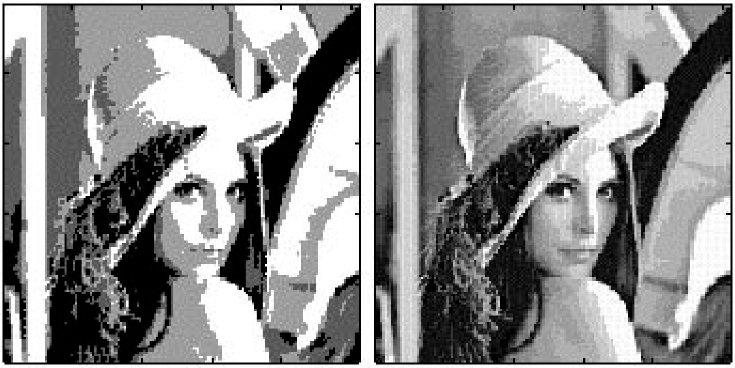


Figure 12.38 – *LBG compression:* $(m = 4, b = 2)$ *on the left,* $(m = 4, b = 4)$ *on the right*

12.16 Digital communications

12.16.1 Introduction

Digital communications offer many challenges to people working in signal processing. The new services provided for cellular communications or the internet pose problems which have to do partly with digital signal processing.

Simply put, "making a digital communication" consists of transmitting a continuous-time signal constructed from a message comprised of a sequence $\{d_k\}$ of bits. To conduct this transmission operation, a device called a *modulator* is used for emitting. When the signal is received, the opposite operation is conducted by a device called a *demodulator*. Therefore, in two-way communications, a modulator and a demodulator are necessary at both ends of the communications line. The word *modem* comes from the contraction of these two words.

According to the characteristic features of the channel, low-pass or high-pass, the modulation is implemented in *baseband*, or on a *carrier frequency*.

Baseband modulation

In *baseband* modulations, the modulation operation consists of producing a signal $x_e(t)$ (Figure 12.39) defined by:

$$x_e(t) = \sum_{k=-\infty}^{+\infty} a_k h_e(t - kT) \tag{12.46}$$

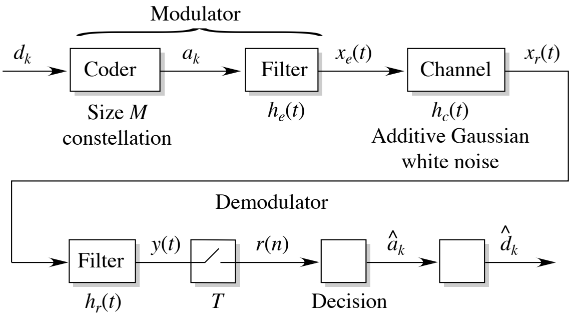


Figure 12.39 – *Baseband modulation and demodulation*

In expression 12.46:

- a_k is a sequence of symbols with possible values in a finite set of M values, called a *constellation*. The sequence a_k is constructed from the sequence of the bits d_k by an coding algorithm that associates a sequence of bits with each symbol of the constellation. Most of the time, the constellation is comprised of $M = 2^N$ real values, and hence, to each possible value in the constellation corresponds a code word with the length:

$$N = \log_2(M) \text{ bits} \tag{12.47}$$

- T represents the time interval between the transmission of two consecutive symbols. The *symbol rate*, or *modulation rate*, expressed in *Bauds*, is defined by:

$$R = \frac{1}{T} \tag{12.48}$$

– The choice of the impulse function $h_e(t)$ depends on the characteristic features of the transmission channel.

"On the other end of the line", the operations that have to be performed to reconstruct the original sequence consist of a filtering, followed by a sampling and a test designed to retrieve the symbols a_k as best as possible. These symbols are then decoded to obtain the sequence of bits d_k. The exercises in this paragraph shed light on the methods used all along the transmission channel.

Carrier frequency modulation

In *carrier frequency* modulations F_0 (Figure 12.40), the transmitted signal has the expression:

$$\begin{aligned} x_e(t) &= \text{Re}\left(\alpha(t)e^{2j\pi F_0 t}\right) \\ &= \text{Re}(\alpha(t))\cos(2\pi F_0 t) - \text{Im}(\alpha(t))\sin(2\pi F_0 t) \end{aligned} \tag{12.49}$$

$$\text{where } \alpha(t) = \sum_{k=-\infty}^{+\infty} a_k h_e(t-kT) \tag{12.50}$$

The symbols a_k are complex. The complex signal $\alpha(t)$ is called the *complex envelope of the real signal $x_e(t)$ with respect to the frequency F_0*. The real and imaginary parts of $\alpha(t)$ are called the *phase component* and the *quadrature component* of $x_e(t)$ respectively.

The complex envelope can be quite useful both in theoretical calculations and in simulation programs. This is due to the fact that the error probabilities, in the presence of additive white noise, do not depend on the choice of the carrier frequency F_0. Therefore, it is useless in a simulation to generate the modulated signal.

The diagram in Figure 12.40 shows an implementation of the modulator for which $h_e(t)$ was assumed to be real. As for the demodulator, the received signal is first processed so as to extract the real and imaginary parts from the complex envelope. Each of the two signals is then filtered and sampled. The two results make up the real and imaginary parts of a sequence of complex observations used by the decision-making system to determine the transmitted sequence of symbols, then the transmitted sequence of bits.

Relation between symbol rate and bit rate

As a result of the operation putting together the bits in groups of N, the time T between the transmission of the two consecutive symbols is equal to N times the time interval T_b between two consecutive bits. If the binary rate is denoted by $D = 1/T_b$, and if we use expressions 12.47 and 12.48, we get formula 12.51,

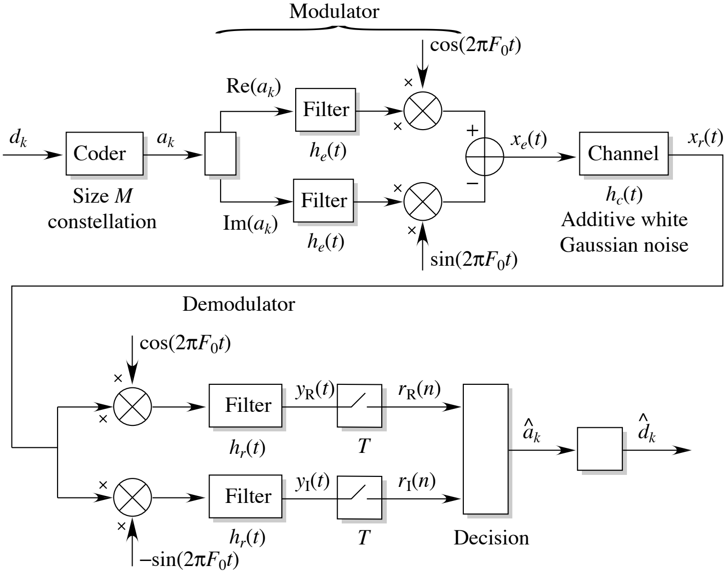


Figure 12.40 – *Carrier frequency modulation and demodulation*

which shows the relation between the *symbol rate*, the *binary rate*, and the *size of the modulation alphabet*:

$$D = R\log_2(M) \tag{12.51}$$

12.16.2 8-phase shift keying (PSK)

We will start with the example of the 8-phase digital modulator that associates the portion of the sinusoidal signal $x(t) = A\cos(2\pi F_0 t + \Phi)$ lasting a duration of T with each 3 bit group. Because there are 8 ways of grouping 3 bits together, the values of Φ are chosen in the set comprised of the 8 phases regularly spread out between 0 and 2π. This set defined by $\{0,\ 2\pi/8,\ 4\pi/8,\ \ldots,\ 14\pi/8\}$ makes up the constellation.

This constellation is represented in Figure 12.41, which shows a coding example. Notice that the chosen coding is such that the codes of two adjacent symbols differ *by only one bit*. This is called a *Gray* code. We will see (equation 12.63) what the point of such a coding is for the value of the bit error probability. Figure 12.42 shows the signal corresponding to a sequence of 15 bits.

Let us check that the complex envelope of a phase modulation is written:

$$\alpha(t) \;=\; A\sum_k a_k \mathrm{rect}(t - kT)$$

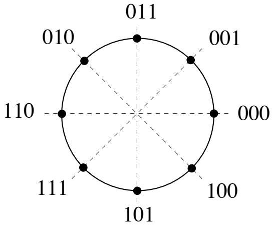


Figure 12.41 – 8-*PSK state constellation*

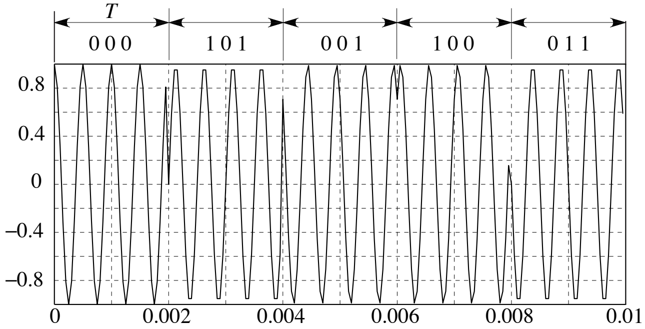


Figure 12.42 – 8-*PSK modulation signal*

where the $a_k = e^{j\Phi_k}$ are the complex symbols shown in Figure 12.41. Indeed, the signal $x_e(t)$ in the interval $(kT, (k+1)T)$ is written:

$$\begin{aligned} x_e(t) &= A\cos(2\pi F_0 t + \Phi_k) = A \times \text{Re}\left(e^{2j\pi F_0 t + j\Phi_k}\right) \\ &= \text{Re}\left(A a_k e^{2j\pi F_0 t}\right) = \text{Re}\left(\alpha(t) e^{2j\pi F_0 t}\right) \end{aligned}$$

By referring to expression 12.49, we can conclude that the complex envelope is equal to Aa_k in the interval $(kT, (k+1)T)$, which is the expected result.

Exercise 12.21 (Phase modulator)
Consider an 8-PSK modulator and a binary rate of 1,500 bps.

1. Determine the symbol rate.
2. Determine a coding such that 2 neighboring points of the constellation differ by only one bit (*Gray code*).
3. Write a program that generates the signal transmitted, for a carrier frequency $F_0 = 2$ kHz (take a sampling frequency equal to 20 kHz for the display) and for the binary sequence [000 101 001 100 011].

12.16.3 PAM modulation

We now return to expression 12.46 of a baseband modulation, and we will consider that the sequence $\{a_k\}$ is a sequence with possible values in the constellation comprised of M real symbols and defined by:

$$a_k \in \{-(M-1) : 2 : +(M-1)\} \tag{12.52}$$

Most of the time, M is a power of 2. For instance, if $M = 8$, the alphabet is the set $\{-7, -5, -3, -1, +1, +3, +5, +7\}$. This modulation is called an M state *pulse amplitude modulation*, or M-PAM. Still in the case where $M = 8$, the association of 3 bit sequences with alphabet symbols can be done by using the following Gray code:

Sequence	000	001	011	010	110	111	101	100
Symbol	-7	-5	-3	-1	$+1$	$+3$	$+5$	$+7$

An example of the type of signal transmitted, when $h_e(t)$ is a rectangular impulse with a duration T, is shown in Figure 12.43.

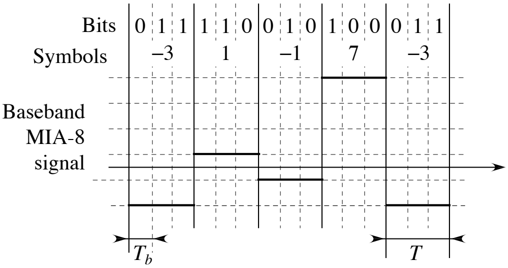


Figure 12.43 – *An example of 8-PAM modulation*

We will see, using formula 12.54, that this signal, in the case where $h_e(t)$ is a rectangular impulse, theoretically takes up an infinite amount of space in the spectrum, meaning that it cannot be transmitted through a B band limited channel without distortion, particularly if $B < 1/T$. We will see in paragraph 12.16.5 that it is then preferable to choose $h_e(t)$ so as to satisfy a criterion better suited to the demodulation problem, called the Nyquist criterion.

In any case, the signal is deformed when it is transmitted through the channel. If we assume that the channel acts as a linear filter with the impulse response $h_c(t)$ added to a noise $w(t)$, the received signal is the following:

$$x_r(t) = \sum_{k=-\infty}^{+\infty} a_k h(t-kT) + w(t)$$

where the impulse $h(t) = (h_e \star h_c)(t)$ corresponds to the cascading of the emission filter $h_e(t)$ and $h_c(t)$.

Under the hypothesis that $w(t)$ is a white, centered, Gaussian noise, it can be shown [50] that in order to perform an optimal detection of the sequence of symbols $\{a_k\}$, and hence of the sequence of binary elements $\{d_k\}$, all we need to do is filter the signal $x_r(t)$ as it is received by the *matched* filter with the impulse response $h^*(-t)$, and to make the decision based only on the samples taken at the symbol rate $1/T$ from the matched filter's output.

Generally speaking, the transmission channel acts as a low-pass filter, which causes the signals to stretch out in time. In most cases, such as for instance the phone channel, $h_c(t)$ stretches out beyond T. As a consequence, the signal corresponding to the symbol a_k "overflows" onto the following time intervals. This is called *InterSymbol Interference*, or *ISI*.

This phenomenon is a nuisance as it makes it more difficult to retrieve the symbols from the matched filter's output samples. Even if we assume that the noise is Gaussian and white, the optimal receiver has to use a complex algorithm, called the *Viterbi algorithm*, to retrieve the most likely sequence of symbols (see paragraph 12.17). However, as we will see, retrieving the symbols is very simple in the absence of ISI.

Error probability and signal-to-noise ratio

An important element for determining the performances parameters of the transmission system is the value of the *Bit Error Rate*, or *BER*. For the simplest modulations, it is possible to find an analytical expression [50]. However, most of the time, it is simply estimated using a simulation program that compares the sequence of emitted bits to the sequence of decided bits. Remember, while we are on the subject, that to obtain a 10% accuracy on the error probability P_e with a 70% confidence, we need a test sequence with a length $N \approx 100/P_e$. For $P_e = 10^{-2}$, this means $N = 10{,}000$, making it understandable that this can lead to a long simulation time, even for a fast computer.

The BER is usually plotted against the signal-to-noise ratio E_b/N_0 between the mean energy E_b necessary to send a bit and the quantity N_0, *where* $N_0/2$ *represents the* psd *of a white noise.* We wish to emphasize that N_0 is expressed in Joules, as it should be, because N_0 represents a power spectral density, and is therefore measured in Watt/Hz. The ratio E_b/N_0 can also be expressed as a function of the signal-to-noise ratio of the powers. If P_s refers to the power of the useful signal, and P_b refers to the noise power in the $(-B, +B)$ band, then:

$$P_s = \frac{E_b}{T_b} \text{ and } P_b = N_0 B$$

Hence the signal-to-noise ratio also has the expression:

$$\frac{E_b}{N_0} = \frac{P_s BT}{P_b} = \frac{P_s}{P_b}\frac{B}{R}$$

where R refers to the symbol rate (expression 12.48).

12.16.4 Spectrum of a digital signal

Consider the digital signal:

$$x(t)=\sum_k a_k g(t-kT+U) \tag{12.53}$$

where $\{a_k\}$ is a *WSS* random sequence with possible values belonging to a finite alphabet. We will use the notations $m_a=\mathbb{E}\{a_n\}$ and $R_a(k)=\mathbb{E}\{a^c_{n+k}a^c_n\}$ where $a^c_n=a_n-m_a$. $g(t)$ is a modulation pulse, and U is a random variable uniformly distributed on $(0,T)$ and independent of the random variables $\{a_k\}$. T represents the time interval separating the transmission of two consecutive symbols. We are going to prove that the signal $x(t)$ is WSS stationary and that its power spectral density has the expression:

$$\begin{aligned} S_x(f) &= \frac{|G(f)|^2}{T}\sum_\ell R_a(\ell)e^{-2j\pi f\ell T} \\ &\quad +\frac{|m_a|^2}{T^2}\sum_{\ell\neq 0}\left|G\left(\frac{\ell}{T}\right)\right|^2\delta(f-\ell/T) \end{aligned} \tag{12.54}$$

First, notice that the probability distribution of U has the probability density $p_U(u)=\mathbb{1}(u\in(0,T))/T$. Hence, because the random variables a_k and U are assumed to be independent (the product's expectation is equal to the the product of the expectations), we have:

$$\mathbb{E}\{x(t)\}=m_a\sum_k\mathbb{E}\{g(t-kT+U)\}=\frac{m_a}{T}\sum_k\int_0^T g(t-kT+u)du$$

Making the variable change $v=t-kT+u$ and noticing that the integrals for $k\in\mathbb{Z}$ are defined on adjacent intervals, we get:

$$\mathbb{E}\{x(t)\}=\frac{m_a}{T}\sum_k\int_{t-kT}^{t-kT+T}g(v)dv=\frac{m_a}{T}\int_{-\infty}^{+\infty}g(v)dv=\frac{m_a}{T}G(0)$$

where $G(f)$ refers to the Fourier transform of $g(t)$.

We now calculate $\mathbb{E}\{x(t+\tau)x^*(t)\}$. Because a_k and U are independent:

$$\mathbb{E}\{x(t+\tau)x^*(t)\}=\sum_k\sum_n\mathbb{E}\{a_k a_n^*\}\,\mathbb{E}\{g(t+\tau-kT+U)g^*(t-nT+U)\}$$

If we make the variable change $v=t+\tau-kT+u$ and use the expression $p_U(u)=\mathbb{1}(u\in(0,T))/T$, we get:

$$\begin{aligned} &\mathbb{E}\{g(t+\tau-kT+U)g^*(t-nT+U)\} \\ &\qquad=\frac{1}{T}\int_0^T g(t+\tau-kT+u)g^*(t-nT+u)du \\ &\qquad=\frac{1}{T}\int_{t-kT}^{t-kT+T}g(v)g^*(v-\tau-(n-k)T)dv \end{aligned}$$

By defining $\ell = (n-k)$, by using the equality $\mathbb{E}\{a_k a_n^*\} = R_a(k-n) + |m_a|^2$ and by noticing that the integrals for $k \in \mathbb{Z}$ are defined on adjacent intervals, we have:

$$\mathbb{E}\{x(t+\tau)x^*(t)\} = \sum_\ell (R_a(\ell) + |m_a|^2)\frac{1}{T}\int_{-\infty}^{+\infty} g(v)g^*(v-\tau+\ell T)dv$$

If we denote by $h(\theta)$ the convolution of $g(\theta)$ with $g^*(-\theta)$, the integral can be written simply as $h(\tau - \ell T)$. This leads us to:

$$\mathbb{E}\{x(t+\tau)x^*(t)\} = \frac{1}{T}\sum_\ell (R_a(\ell) + |m_a|^2)h(\tau - \ell T)$$

which depends only on τ. $x(t)$ is therefore a WSS process. The Fourier transform of $h(\tau - \ell T)$ is $H(f)\exp(-2j\pi\ell Tf)$. Because $h(\theta) = g(\theta) \star g^*(-\theta)$, we have $H(f) = G(f)G^*(f) = |G(f)|^2$. If we use the Poisson formula 2.4, we infer that:

$$\sum_\ell h(\tau - \ell T) = \frac{1}{T}\sum_m H(m/T)\exp(+2j\pi m\tau/T)$$

If we calculate the Fourier transform of the two sides of $\mathbb{E}\{x(t+\tau)x^*(t)\}$, we then get:

$$\Gamma(f) = \frac{|G(f)|^2}{T}\sum_\ell R_a(\ell)\exp(-2j\pi f\ell T) + \frac{|m_a|^2}{T^2}\sum_m |G(m/T)|^2\delta(f - m/T)$$

where $\delta(f - f_0)$ was used to denote the Fourier transform of $\exp(2j\pi f_0\tau)$. Remember that the psd is precisely $\Gamma(f)$ from which the peak $|\mathbb{E}\{x(t)\}|^2\delta(f)$ at the origin is subtracted, which leads us to the expected result.

As you can see, the spectrum depends, on the one hand on the chosen pulse, and on the other hand on the correlations introduced in the sequence $\{a_k\}$ by way of the expression:

$$S_a(f) = \sum_\ell R_a(\ell)e^{-2j\pi f\ell T}$$

Notice that this function is periodic with period $1/T$. Therefore all we have to do is calculate it on an frequency interval with a length of $1/T$, or by considering the normalized variable $u = fT$, on an interval with a length of 1. If the sequence $\{a_n\}$ is real, then $S_a(f)$ is an even function, and we can restrict our calculations to the positive frequencies. Theoretically, $S_x(f)$ has an infinite support. However, because of the multiplication of the periodic function $S_a(f)$ by the function $|G(f)|^2$, we can restrict the plotting of $S_x(f)$ to a few length $1/T$ intervals, since $|G(f)|^2$ usually decreases fast, typically like $1/f^2$ for the rectangular pulse. This is why in exercise 12.23, we only represented the function in the frequency interval $(0, 1/T)$, that is to say the interval $(0, 1)$ for the normalized variable $u = fT$.

Peaks can appear in multiples of $1/T$ in the case where $m_a \neq 0$. These peaks can be used to retrieve the symbol rate by filtering the signal after receiving it.

In the particular case where the sequence a_k is a sequence of uncorrelated centered variables with the same variance σ_a^2, $R_a(k) = \mathbb{E}\{a_{n+k}a_n\} = 0$ for $k \neq 0$ and the spectrum's expression comes down to:

$$S_x(f) = \sigma_a^2 \frac{|G(f)|^2}{T} \tag{12.55}$$

In the following exercises, we are going to study, through calculation, then through simulation the coding contribution for two codes of great practical importance: the AMI code and the HDB3 code. We will see in particular that these codes are such that $S_x(f)$ is null in $f = 0$. This is one of the important elements involved in the choice of a modulation, because many systems "pass" the spectrum's components very poorly around 0. Also, a zero gain in 0 can possibly add a continuous components, ensuring the system's power supply.

Exercise 12.22 (AMI code)

In AMI coding, the sequence a_k is obtained using the following coding rule: if the bit $d_k = 0$, then $a_k = 0$ is transmitted, and if the bit $d_k = 1$, then $a_k = -1$ and $a_k = 1$ are *alternately transmitted*. AMI stands for Alternate Mark Inversion. We can check that $a_k \in \{-1, 0, 1\}$ is obtained from the $d_k \in \{0, 1\}$ by using the following relations:

$$\begin{cases} s_{k+1} = (1 - 2d_k)s_k \\ a_k = d_k s_k \end{cases}$$

1. Let $\{d_k\}$ be an sequence of random variables, i.i.d. in $\{0, 1\}$, with the probability $\Pr(d_k = 0) = \Pr(d_k = 1) = 1/2$. Calculate $\mathbb{E}\{a_k\}$ and $\mathbb{E}\{a_{n+k}a_n\}$. Use the result to find the expression:

   $$S_a(f) = \textstyle\sum_k R_a(k)e^{-2j\pi kfT}$$

2. Design a program that calculates the theoretical psd of $S_a(f)$ and compares it with the psd estimate obtained through simulation (use the `welch` function from exercise 9.1 page 327 to estimate the psd).

Exercise 12.23 (HDB3 code)

In AMI coding, see exercise 12.23, the presence of a long sequence of zeros can cause the receiver to desynchronize. We then have to make sure never to transmit more than three consecutive zeros. In HDB3 coding (HDB stands for High Density Bipolar) solves this problem in the following way: when we encounter a sequence of four consecutive zeros, the fourth zero is coded as a

"1". To prevent any ambiguity when decoding, this 1 is coded by violating the alternation rule: this is called *bipolar violation*.

EXAMPLE: consider the sequence 101100000000000010. Its coding leads to:

bit:	+1	0	+1	+1	0	0	0	0	0	0	0	0	0	0	0	0	+1	0
symb.:	+1	0	−1	+1	0	0	0	+1	0	0	0	+1	0	0	0	+1	−1	0

Locally, this sequence has a mean different from 0. To avoid this local decentering phenomenon, an alternating rule is applied to the bipolar violation. In order to do this, we have to introduce a additional variable (p_v) that memorizes the bipolar violation.

If p_1 denotes the variable used to store the polarity of the last bit coded as "1", then we have the following algorithm (see the diagram in Figure 12.44):

1. If the last bit coded as 1 is transmitted with a + polarity, which is memorized as $p_1 = +1$, then the sequence 0000 is associated with either the sequence $0\,0\,0\,+1$, or the sequence $-1\,0\,0\,-1$, depending on whether the bit $p_v = -1$ or $+1$ respectively. Then the polarity of p_v is changed, and we redefine $p_1 = p_v$.

2. In the other case, $p_1 = -1$, the sequence 0000 is associated with either $+1\,0\,0\,+1$, or $0\,0\,0\,-1$, depending on whether the bit $p_v = -1$ or $+1$ respectively. Then the polarity of p_v is changed, and we redefine $p_1 = p_v$.

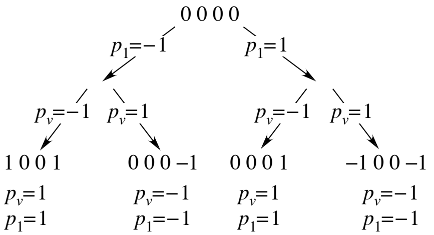


Figure 12.44 – *HDB3 coding: processing the four consecutive zeros and updating the bipolar violation bits*

It can be shown that the resulting sequence is centered. The decoding is particularly simple: if a bipolar violation is encountered, the last 4 bits are set to zero.

1. Starting with the initial conditions $p_v = +1$ and $p_1 = -1$, apply HDB3 coding to the binary sequence:

   0 1 1 1 0 0 0 0 1 0 0 0 0 0 1 0

2. Write a MATLAB® function that performs the HDB3 coding of any sequence of bits.

3. With the use of the `welch` function from exercise 9.1 on page 327, perform the signal's spectrum estimation based on a sequence of 10,000 symbols and for a rectangular pulse with a duration equal to the symbol rate.

Exercise 12.24 (Linear equalization of a communications channel) Consider the double side band modulation with the complex envelope:

$$\alpha(t) = \sum_k a_k h_e(t-kT)$$

The carrier frequency value F_0 is not specified since all of the processings will apply to the complex envelopes of the signals encountered in different places along the transmission line.

Let $a_k = u_k + jv_k$ where the possible values of u_k and v_k belong to $\{-3,-1,+1,+3\}$. This type of modulation is called Quadrature Amplitude Modulation (QAM).

The received signal is sampled at a rate $1/T$. Let $\{x_k\}$ be the sequence of the samples. We will assume that the channel introduces by filtering an intersymbol interference affecting $L=2$ symbols, meaning that:

$$x_k = h_0 a_k + h_1 a_{k-1} + w_k$$

where w_k refers to a white, centered, complex, Gaussian, additive noise. A simple, but not very effective idea is to invert the filter $H(z) = h_0 + h_1 z^{-1}$.

1. Represent the constellation associated with this code. Find a Gray code associated with this constellation.

2. Determine a causal approximation of $G(z) = 1/H(z)$ using a 21 coefficient FIR filter in the following two cases: $h_0 = 1, h_1 = -0.6$ and $h_0 = 1, h_1 = -1.6$.

3. Write a program that represents, in the complex plane, the samples that are emitted, received, and processed by the inverse filter .

12.16.5 The Nyquist criterion in digital communications

Expressing the Nyquist criterion for PAM modulation

Given the digital signal $\sum_k a_k h_e(t-kT)$, corresponding to a PAM modulation, we are going to try to determine the conditions that the pulse $h_e(t)$ has to satisfy so that the emitted signal, once it has travelled through a *B band limited ideal low-pass channel* (its complex gain has the expression $H_c(f) = \mathbb{1}(-B \leq f \leq$

B)), then gone through the matched filter with the impulse response $h_e^*(-t)$, leads an output with *no intersymbol interference at the sampling times.*

Since the channel is B band, we can choose a B band limited signal without being any less general. In that case the received signal can be written:

$$x_r(t) = \sum_k a_k h_e(t - kT) + w(t)$$

where a_k refers to a sequence of symbols from the alphabet $\{-(M - 1), \cdots, -3, -1, +1, +3, \cdots, +(M-1)\}$. If we assume that $p(t) = h_e(t) \star h_e^*(-t)$, and calculate the Fourier transform, we get:

$$P(f) = |H_e(f)|^2$$

Hence, if $P(f)$ is B band limited, that is if $P(f) = 0$ for $|f| > B$, then $h_e(t)$ is B band limited itself. This condition precisely expresses the constraint that has to be satisfied if the channel is B band limited. If $y(t)$ now refers to the output signal of the matched filter[6] with the impulse response $h_e^*(-t)$, then we can write:

$$y(t) = \sum_k a_k p(t - kT) + b(t)$$

where $b(t)$ represents the noise $w(t)$ after it has been filtered. The samples taken from the sampler's output at the times nT then have the expression:

$$\begin{aligned} y(nT) &= \sum_k a_k p(nT - kT) + b(nT) \\ &= a_n p(0) + \underbrace{\sum_{k \neq n} a_k p((n-k)T)}_{\text{ISI}} + b(nT) \end{aligned} \qquad (12.56)$$

The first term is directly related to the symbol a_n emitted at the time nT. The second one represents the contribution, at the sampling times, of all the symbols emitted other than a_n. This term is called the *InterSymbol Interference.*

A situation of particular practical importance is the one where $p(t)$ satisfies the two following conditions, called the *Nyquist criterion*:

1. $P(f) = 0$ for $|f| > B$, where B refers to the channel's bandwidth;

2. $p(kT) = 0$ for $k \neq 0$.

[6] All of the results in this paragraph are still true in the case of side band modulations with the carrier frequency F_0 if we replace $x_r(t)$ with its complex envelope with respect to F_0.

According to the Poisson formula (page 53), condition 2 is equivalent to:

$$\sum_{k=-\infty}^{+\infty} P\left(f-\frac{k}{T}\right) = T\sum_{m=-\infty}^{+\infty} p(mT)\exp(-2\pi jmfT) = Tp(0)$$

This expression means that the algebraic sum of the spectra shifted by $1/T$, $2/T\ldots$ is equal to a constant. If we introduce the symbol rate $R = 1/T$, we can determine a necessary condition for the Nyquist criterion to be satisfied on a B band channel. This condition is expressed:

$$B > \frac{R}{2} \tag{12.57}$$

Let us examine how the Nyquist criterion leads to a simplified decision rule. If $p(t)$ satisfies the Nyquist criterion, then the expression 12.56 giving $y(nT)$ can be simplified and we have:

$$y(nT) = a_n p(0) + b(nT) \tag{12.58}$$

Because $y(nT)$ only depends on one symbol, and because it can be shown (you can do it as an exercise) that the noise samples are independent, since they are uncorrelated and Gaussian, the decision can be taken *symbol by symbol* simply by comparing $y(nT)$ to thresholds.

What we did is start out with a transmission system designed for a pulse $h_e(t)$ such that $p(t) = h_e(t) \star h_e^*(-t)$ verifies the Nyquist criterion, and in the end, the reception set is composed of:

- a matched filter with the impulse response $h_e^*(-t)$;
- a sampler at the rate T;
- and a symbol-by-symbol decision-making system that compares the sample with thresholds.

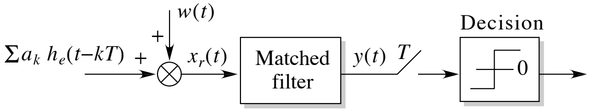


Figure 12.45 – *Diagram of the receiver*

For example, with an PAM modulation with 4 symbols $\{-3,-1,+1,+3\}$, the decision rule is as follows:

Observation	Decision
$y(nT) < -2p(0)$	$\hat{a}_n = -3$
$-2p(0) \leq y(nT) < 0$	$\hat{a}_n = -1$
$0 \leq y(nT) < 2p(0)$	$\hat{a}_n = +1$
$2p(0) \leq y(nT)$	$\hat{a}_n = +3$

When the ISI is low but not completely non-existent, we can still use symbol-by-symbol decision. To explain what a low ISI means, we have to go back to expression 12.56. The most unfavorable case concerning the use of symbol-by-symbol decision occurs when all the symbols other than a_n interfere destructively with the amplitude $a_n p(0)$. For example, if $a_n = 1$ and $p(0) > 0$, all the other symbols assume the value $\pm(M-1)$ causing each contribution to be negative. This leads us to the following definition on how to measure the level of ISI, also called *the maximum ISI*:

$$D = \frac{(M-1)\sum_{k\neq 0}|p(k)|}{|p(0)|} \tag{12.59}$$

If D is close to 0, the ISI is low and the symbol-per-symbol detector will yield excellent results. Otherwise, optimal processing requires a more complex algorithm called the Viterbi algorithm (see page 562) that takes into account all of the received symbols. In paragraph 12.16.6, we will discuss another tool for evaluating the ISI called the *eye pattern*.

The raised cosine Nyquist function

We are now going to define a set of real functions $h_e(t)$ depending on one parameter, and such that $p(t) = h_e(t) \star h_e(-t)$ satisfies the Nyquist criterion. These functions play a major practical role. Consider the function:

$$h_e(t) = \frac{1}{\pi\sqrt{T}} \frac{4\alpha\dfrac{t}{T}\cos\left((1+\alpha)\pi\dfrac{t}{T}\right) + \sin\left((1-\alpha)\pi\dfrac{t}{T}\right)}{\dfrac{t}{T}\left(1-16\alpha^2\dfrac{t^2}{T^2}\right)} \tag{12.60}$$

where the parameter $\alpha \in (0,1)$ is called the *roll-off factor*.

It can be shown by way of a long calculation that $p(t) = h_e(t) \star h_e(-t)$ satisfies the Nyquist conditions, meaning that $p(kT) = 0$ for $k \neq 0$ and $P(f) = 0$ for $|f| > B$ with:

$$B = \frac{1}{2T}(1+\alpha) \tag{12.61}$$

Theoretically, $h_e(t)$ has an infinite duration. However, the function quickly becomes evanescent, and we can maintain satisfactory properties if we truncate the function to keep about a dozen lobes around 0.

The `racnyq.m` function generates the samples of $h_e(t)$ based on the roll-off factor `alpha`, the number of lobes `nblobes` remaining to the right and left of 0, as well as the number of samples `Npts` on the interval with a duration T. Save this function as `racnyq.m`:

```
function hracNyq=racnyq(alpha,nblobes,Npts);
%%====================================================%
%% RACNYQ: Square root raised cosine response         %
%% SYNOPSIS: hracNyq=RACNYQ(alpha,nblobes,Npts)       %
%%     alpha   = Roll-off                             %
%%     nblobes = Response length (even symbol number) %
%%     Npts    = Number of points per symbol          %
%%     hracNyq = Response                             %
%%====================================================%
deminblobes=nblobes/2; tsurT=(1:deminblobes*Npts)/Npts;
a4tsurT=4*alpha*tsurT;
gamma1=pi*(1+alpha)*tsurT; gamma2=pi*(1-alpha)*tsurT;
%=====
num= a4tsurT  .* cos(gamma1) + sin(gamma2);
den=(1- a4tsurT .* a4tsurT) .* tsurT;
%===== den(t0)=0 if tsurT=1/(4*alpha)
t0=find(abs(den) <=  2.2204e-14);
lh=length(num);
if isempty(t0)
   hracNyq=num ./ den;
else
   C1=pi*(1+alpha)/(4*alpha);
   hnul=(-0.5*cos(C1)+(pi/4)*sin(C1))*4*alpha;
   hracNyq=[num(1:t0-1) ./ den(1:t0-1) hnul ...
         num(t0+1:lh) ./ den(t0+1:lh)];
end
h0=4*alpha+pi*(1-alpha);
hracNyq=[hracNyq(lh:-1:1) h0 hracNyq]';
hracNyq=hracNyq/norm(hracNyq);
return
```

A long but not at all difficult calculation shows that:

$$P(f) = \begin{cases} T & \text{for } |f| < \dfrac{1-\alpha}{2T} \\ \dfrac{T}{2}\left[1 - \sin\left(\dfrac{\pi T}{\alpha}(f - 1/2T)\right)\right] & \text{for } \dfrac{1-\alpha}{2T} < |f| < \dfrac{1+\alpha}{2T} \\ 0 & \text{for } |f| > \dfrac{1+\alpha}{2T} \end{cases} \qquad (12.62)$$

with $\alpha \in (0, 1)$. $P(f)$ is called a *raised cosine* function.

Let us now check numerically that the function $p(t)$ satisfies the Nyquist criterion. Let $R = 1/T = 1{,}000$ symbols per second and $B = 600$ Hz (this way we have $B > R/2$). Hence, according to 12.61, $\alpha = 2B/R-1 = 0.2$. To perform the computation, we will take 10 points per symbol time T and truncate $h_e(t)$ down to about 20 lobes around 0. The convolution $p(t) = h_e(t) \star h_e(-t)$ is obtained using the MATLAB® function `conv`. $p(t)$ and its Fourier transform are plotted for 1,024 frequency points. To do this, type:

```
%===== CRITNYQ.M
clear
R=1000; T=1/R; B=600;
%=====
alpha=(2*B/R-1);
NpS=10; Fe=NpS*R; Te=1/Fe; nblobes=20;
h=racnyq(alpha,nblobes,NpS); lh=length(h);
p=conv(h,h(lh:-1:1));
%===== Temporal response
lpm1=length(p)-1;
subplot(211); plot(Te*(-lpm1/2:lpm1/2),p)
set(gca,'xtick',(-30:30)*T)
grid ; set(gca,'xlim',[-5*T 5*T]);
%===== Spectral response
Lfft=1024; fq=Fe*(0:Lfft-1)/Lfft;
subplot(212); plot(fq,abs(fft(p,Lfft))); grid
set(gca,'xlim',[0 R]);
B=(1+alpha)*R/2; text(B,2,'B');
```

The results are shown in Figure 12.46. As you can see, the function $p(t)$ is null for every multiple of T, except at the origin, and the spectrum is B band limited, meaning that $P(f) = 0$ for $|f| > B = (1+\alpha)/2T$.

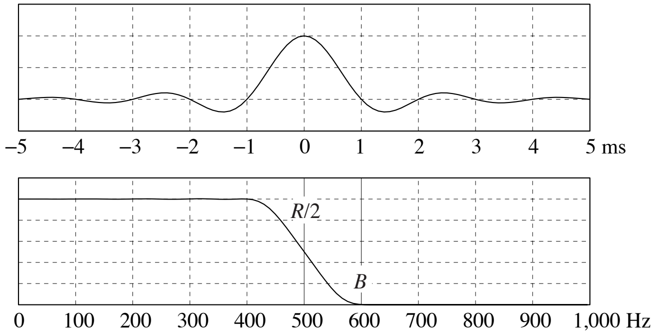


Figure 12.46 – *A raised cosine function satisfying the Nyquist criterion*

12.16.6 The eye pattern

As we have said before, the receiver shown in Figure 12.45 is so simple because of the absence of ISI. This is why it is essential to have access to a practical tool for measuring its level. We have already given on page 549 a quantitative definition, with equation 12.59, of the maximum ISI, and we are now going to present another very important tool, called the *eye pattern*, for reasons that will be obvious if you refer to Figure 12.47.

The eye pattern is obtained by superimposing a large number of trajectories, with durations $2T$, of the matched filter's output signal. It can be displayed on an oscilloscope by synchronizing with the symbol rate $1/T$. The afterglow of the screen makes it possible for the superposition to persist. In the presence of noise, as we are going to see, the wider the eye is vertically, the lower the error probability. Therefore, we have to choose the decision time where the eye is vertically "widest".

In the following exercise, we are going to generate a complete simulation line for 2-PAM modulation, in the form of five different programs. We will examine the eye pattern and choose the optimal sampling time. All the computed values are saved as we go along, so they can be used as input data for the next program.

Exercise 12.25 (2-PAM modulation)
Consider a PAM modulation with two levels $\{-1,+1\}$. The binary rate is equal to 1,000 bps. The signal's expression is:

$$x_e(t) = \sum_k a_k h_e(t-kT) = \sum_k a_k \mathrm{rect}(t-kT)$$

To display the signal as if it were time-continuous, choose $F_s = 20$ kHz as the sampling frequency.

1. Write a program that computes the samples `xe` of the signal $x_e(t)$ for a random sequence of 300 bits. Display part of the chronogram of `xe` (`zoom xon`). Save all of the values.

2. Write a program that generates and displays the received signal `xr`. To simulate a transmission channel, use a low-pass filter $h_c(t)$ the impulse response of which is obtained by `hc=rif(lhsT*NT-1,bc)` with `lhsT=3.5` and `bc=0.06`, and where `NT` refers to the number of points corresponding to the time interval between two symbols. Therefore the channel's response stretches out over 3.5 symbols.

3. Let $h(t) = (h_c \star h_e)(t)$. The receiver's matched filter therefore has the impulse response $h(-t)$. The matched filter's output signal is denoted by `xa`. Write a program that superimposes the "sections" of `xa` lasting a duration of two "symbol periods" so as to display the *eye pattern*. The

program will have to be designed so that the time coordinate of the place where the eye is vertically "widest" can be interactively defined as input data. You can use the function `ginput`.

4. Write a program that displays the matched filter's output signal `xa` and superimposes the values taken from the sampler's output.

5. Write a program that, based on the value of the signal-to-noise ratio, expressed in dB:

   – adds a white, centered, Gaussian noise with the power P_b to the received signal `xc`;

   – decides that the bit is equal to 1 if its value at the sampler's output is positive and 0 if it is not;

   – and evaluates the error probability.

In exercise 12.25, the overall impulse response ensures that the ISI is low enough for the symbol-by-symbol decision to yield good results. The low level of interference is clearly shown by the eye pattern without noise represented in Figure 12.47. The trajectories almost converge to the same point at multiples of T. Therefore, if the sampling is done at these times, the values located around 1 are likely to correspond to the transmission of a bit 1.

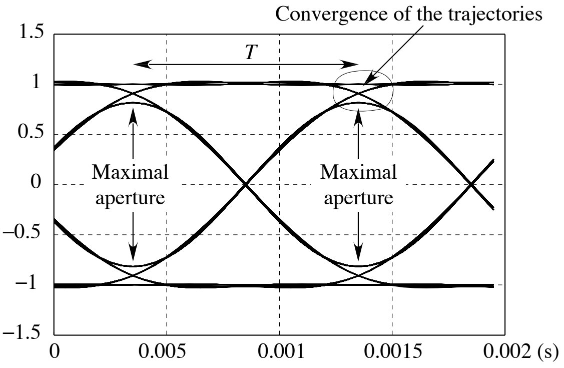


Figure 12.47 – *PAM-2: eye pattern without noise*

12.16.7 PAM modulation on the Nyquist channel

We are now going to present a sequence of programs designed to simulate a PAM modulation on the Nyquist channel. The following paragraphs describe, in this order:

– the generation of symbol sequences;

- the generation of the emitted digital signal;
- the addition of noise by the channel;
- matched filtering and the examination of the eye pattern;
- symbol-by-symbol decision.

Generating the sequence of symbols

The `gsymb.m` function generates a sequence of N symbols taken from the alphabet $\{-(M-1), \cdots, -3, -1, +1, +3, \cdots, (M-1)\}$ with $M = 2^m$. Save this function as `gsymb.m`:

```
function [symb,Esymb]=gsymb(m,N)
%%==================================================%
%% Generates a random sequence with values in an MIA  %
%% M=2^m symbols alphabet with a uniform distribution %
%% SYNOPSIS: [symb,Esymb]=GSYMB(m,N)                  %
%%    m    such as M=2^m                              %
%%    N    = Sequence length                          %
%%    symb = Random sequence                          %
%%    Esymb = Mean energy                             %
%%==================================================%
M=2^m; alphabet=2*(0:M-1)+1-M;
la=length(alphabet);
ind=fix(rand(1,N)*la)+1;
symb=alphabet(ind);
Esymb=alphabet*alphabet'/M;
return
```

Generating the receiver filter's output signal

We are going to try to numerically compute the sample at the rate $T_s = T/N_T$ of the signal $x_e(t) = \sum_k a_k h_e(t - kT)$. N_T represents the number of samples per symbol duration T. These samples can be used for creating the continuous-time signal using a digital-to-analog converter. In this case, these samples will be used for the display. They have the expression:

$$x_e(nT_s) = \sum_k a_k h_e(nT_s - kN_T T_s) = \sum_k a_k h_e((n - kN_T)T_s)$$

The samples of $h_e(t)$ taken at the rate T_s can be obtained with the use of the `racnyq.m`function. The function $h_e(t)$ can be truncated down to about 30 lobes around 0. If we choose a high enough value for N_T, the resulting plot of $x_e(t)$ is "almost continuous". To calculate the samples of $x_e(t)$, we can use the diagram explaining the principle, from Figure 12.48, drawn for $N_T = 4$. If we let $n = \ell N_T + r$, where $r = 0, \ldots, N_T - 1$, then we indeed have:

$$x_e((\ell N_T + r)T_s) \;=\; \sum_k a_k h_e(((\ell - k)N_T + r)T_s)$$

If we let $\tilde{x}_r(\ell) = x_e((\ell N_T + r)T_s)$ and $\tilde{h}_r(\ell) = h_e((\ell N_T + r)T_s)$, then we can write:

$$\tilde{x}_r(\ell) \;=\; \sum_k a_k \tilde{h}_r(\ell - k)$$

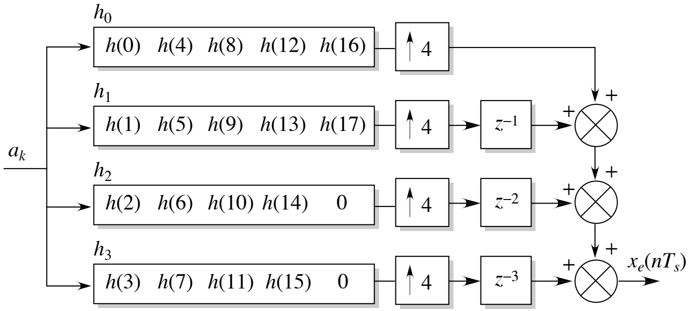


Figure 12.48 – *Constructing the signal transmitted based on the symbols and the polyphase components of the emission filter in the case where $N_T = 4$*

The component $\tilde{x}_r(\ell)$ is called the r-th *N_T-polyphase component.* It is obtained by filtering the sequence a_k by the filter with the impulse response $\tilde{h}_r(\ell)$ obtained by undersampling the sequence $h_e(mT_s)$ by a factor N_T. In MATLAB®, given a sequence `he` of the coefficients of $h_e(t)$, the filter with the impulse response $\tilde{h}_r(\ell)$ is obtained simply by typing:

```
hr = he(r:NT:length(he));
```

In Figure 12.48, we chose a response $h_e(t)$ with 18 non-zero coefficients. The number of samples per symbol is $N_T = 4$. The transmitted signal is obtained by superimposing the 4 shifted components.

The roll-off expression of α is obtained with relations 12.51 and 12.61:

$$\alpha = 2\frac{B}{R} - 1 = 2\frac{B}{\log_2(M)D} - 1$$

Once the samples of $x_e(t)$ have been generated, a noise is added by setting the signal-to-noise ratio. The noised signal is filtered by the filter with the impulse response $h_e(-t)$. $y(t)$ is the resulting signal. The following program computes the samples of $y(t)$ and displays the result of the superimposing the trajectories with a duration of $2T$:

```
%===== OEILNYQ.M
% Eye pattern for a Nyquist Channel
% Uses: GSYMB, RACNYQ, GSIG
clear
N=5000;          % Sequence length (bits) of the source
NbTraj=200;      % Number of trajectories
%===== Binary rate
Db=1000; tt=sprintf('Binary rate: %4.0f bits/s',Db);
disp(tt); petitm=input('Number of bits per symbol: ');
M=2^petitm;
%===== Modulation alphabet
alphabet=2*(0:M-1)+1-M;
%===== Baud rate
Dsymb=Db/petitm;
Bmin=Dsymb/2;                % alpha = 0
Bmax=Dsymb;                  % alpha = 1
tt=sprintf('Band of the channel B between %3.1f and %3.1f :',...
            Bmin,Bmax);
disp(tt); Bc=input('B = ');
if (Bc<Bmin)
   disp('ISI=0 impossible: you must increase the bandwidth.');
   return;
end
if (Bc>Bmax),
   disp('The bandwidth must be decreased.'); return;
end
alpha=(2*Bc/Dsymb)-1; nblobes=30;
[symb, Esymb]=gsymb(petitm,N);        % Generation of the symbols
%===== Square root Nyquist
NpS=10; he=racnyq(alpha,nblobes,NpS);
lh=length(he); xe=zeros(NpS,N);
%===== Emitted signal (Polyphase components in the columns)
for ii=1:NpS,hr=he(ii:NpS:lh); xe(ii,:)=filter(hr,1,symb); end
xe=xe(:,1:N); st=zeros(NpS*N,1); st(:)=xe;
%===== Mean energy per symble
Es_sim=(st'*st)/N; % Estimated value
Es=Esymb*(he'*he);
%===== Mean energy per bit
Eb=Es/petitm; RSB=input('Ratio Eb/N0 (en dB) = ');
PB=Eb*10^(-RSB/10); sigmab=sqrt(PB/2);
xt=st+sigmab*randn(NpS*N,1); % Addition of the noise
%===== Square root Nyquist
htfa=he(lh:-1:1);            % Matched filter
stfa=filter(htfa,1,xt);      % Output to be sampled
%===== Eye pattern
nbtraces=(N-nblobes)/2;
soeil=zeros(NpS*2,nbtraces); soeil(:)=stfa(nblobes*NpS+1:N*NpS);
nivMax=M-1+4*sigmab+4; t0=NpS+1;
```

```
plot([t0 t0],[-nivMax nivMax],':'); hold on
%===== Some trajectories
for tt=1:NbTraj, plot((1:2*NpS),soeil(:,tt)); end
%===== Sampling time
t0=NpS+1; plot([t0 t0],[-nivMax nivMax],':'); hold off
set(gca,'xlim',[1 2*NpS])
save diagoeil
```

By superimposing portions of trajectories with a duration $2T$, that is to say $2N_T$ points, we get the eye pattern shown in Figure 12.49. If M refers to the alphabet size, then there are $(M-1)$ eye apertures. In the case of a non-ideal channel, the trajectories no longer have the shapes shown here. In particular, they no longer perfectly converge in at the multiples of T, which is imposed by the Nyquist criterion.

Figure 12.49 represents the eye pattern for $M = 4$, a rate of 1,000 bps, and a spectrum support of 300 Hz, hence $\alpha = 0.2$.

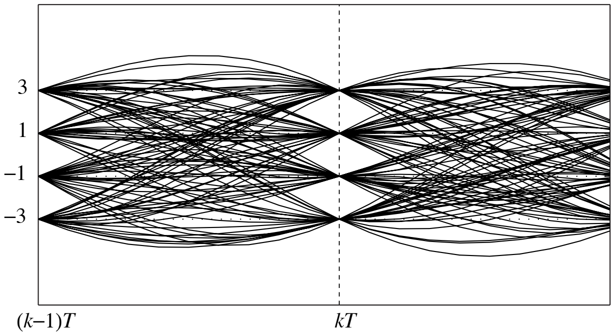


Figure 12.49 – *The eye pattern for $M = 4$. The noise is null. Spectrum support $B = 300$ Hz, rate 1,000 bps*

Figure 12.50 shows the same eye diagram for $M = 4$, but for a spectrum support of 500 Hz, hence $\alpha = 1$. The vertical opening at the times kT is the same as before. In terms of probability, the performances are the same. However, if the sampling time is slightly shifted, the vertical opening becomes wider at that time for the 500 Hz. The error probability will be smaller. To put it more simply, the vertical opening guarantees a better resistance when the sampler is desynchronized.

Figures 12.51 and 12.52 show the eye pattern for a signal-to-noise ratio of 15 dB.

If we "place ourselves" in the center of the eye, and set the thresholds to the values -2, 0 and 2, the error probability has to remain very small, which is checked with the following program. Type:

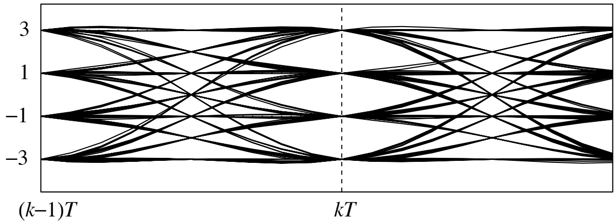


Figure 12.50 – *Eye pattern for* $M = 4$. *The noise is null. Spectrum support* $B = 500$ *Hz, rate 1,000 bps*

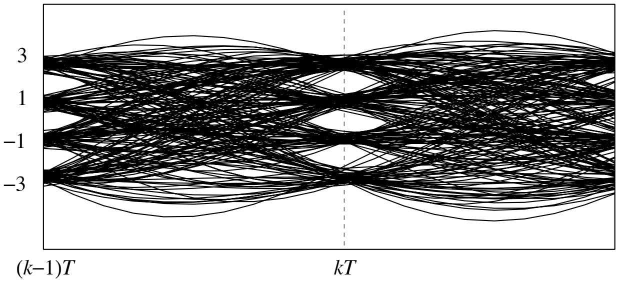


Figure 12.51 – *Eye pattern for* $M = 4$. *The signal-to-noise ratio is equal to* 15 *dB, the spectrum support is* $B = 300$ *Hz and the rate 1,000 bps*

```
%===== PEESTIM.M
% Sampling the output of the matched filter
clear; load diagoeil
%===== Thresholds
seuil=2*(0:M-2)+1-M+1; seuil=[-inf seuil inf];
stfaech=stfa(t0:NpS:N*NpS); lsfa=length(stfaech);
%===== Decision symb/symb by the "nearest" rule
aux=stfaech(NpS:lsfa); laux=length(aux);
deci=zeros(1,laux); lalphabet=length(alphabet);
```

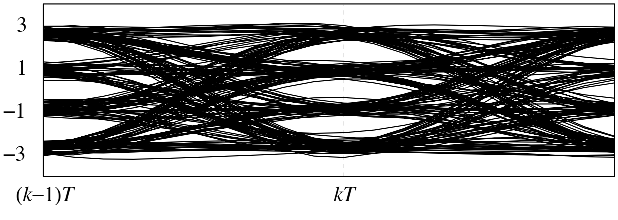


Figure 12.52 – *Eye pattern for* $M = 4$. *The signal-to-noise ratio is equal to* 15 *dB, the spectrum support is* $B = 500$ *Hz and the rate 1,000 bps*

```
for jj=1:lalphabet
    v=find(aux>seuil(jj) & aux<=seuil(jj+1));
    deci(v)=alphabet(jj)*ones(1,length(v));
end
decal=nblobes-NpS; deci=deci(decal+1:length(deci));
ldiff=length(deci); diff=deci - symb(1:ldiff);
nbe=length(find(diff~=0)); pe=nbe/length(diff);
TEB=pe/petitm;
disp('*********************************************')
tt=sprintf('*Binary rate: %d bits/s',Db); disp(tt)
tt=sprintf('*Symbol rate: %3.1f bauds, M=%i',Dsymb,M);
disp(tt)
tt=sprintf('*Channel band: %3.0f Hz, alpha: %3.1f',Bc,alpha);
disp(tt)
tt=sprintf('*Eb/N0=%ddB       ==> TEB=%5.2d ', RSB,TEB);
disp(tt)
tt=sprintf('*Nb errors/symbol= %i pour N=%i',nbe,N); disp(tt)
disp('*********************************************')
```

The program estimates the *error probability P_s per symbol*. If *the signal-to-noise ratio is high*, we can consider that the only errors are caused by a decision in favor of one of the two symbols adjacent to the symbol actually transmitted. Therefore, if we are using a Gray code, these adjacent symbols only cause one *false bit over the $m = \log_2(M)$ emitted bits*, leading us to the following formula, which gives us the relation between the probability per symbol P_s and the *bit error rate* "BER":

$$\text{BER} \approx \frac{P_s}{\log_2(M)} \tag{12.63}$$

With this program, we can also test the error probability increase when the sampling time is shifted. All we need to do is modify the variable `t0`. We have to check that this increase becomes greater as the chosen band becomes narrower. However, when we are right in the center of the eye (where the trajectories converge in the absence of noise), the error probability is independent of the value of B.

12.17 Linear equalization and the Viterbi algorithm

Consider the case of the *binary baseband modulation* that associates the continuous-time signal $x_e(t) = \sum_k a_k h_e(t - kT)$ with the sequence $\{d_k\}$ of binary elements $\{0, 1\}$ according to the following rule: if $d_k = 0$, then $a_k = -1$ and if $d_k = 1$, then $a_k = 1$. The signal $h_e(t)$ refers to the impulse response of the emission filter. The signal $x_e(t)$ is filtered by the transmission channel and is subjected to additive noise. The received signal can then be written

$x_r(t) = \sum_k a_k h(t - kT) + w(t)$ where $h(t)$ is obtained by convoluting the filter $h_e(t - kT)$ with the channel filter, and $w(t)$ is the noise. When the signal is received, it is filtered by the matched filter then sampled. The result is a sequence of values $x_a(n)$, the expressions of which linearly depend on the sequence a_k of the type:

$$x_a(n) = \sum_k a(k)g(n-k) + b(n) \tag{12.64}$$

The term $b(n)$ accounts for the noise, which is assumed to be Gaussian. Without being any less general, we can assume that $b(n)$ is white. If it is not, we know that there exists a causal filter that can make it white. This filter changes the values of the coefficients $g(k)$, but the general form of expression 12.64 stays the same. The variance of $b(n)$ is denoted by σ_b^2.

The coefficients $g(n)$ take into account the emission filter, as well as the channel filter and the receiver matched filter. From now on, the sequence $g(n)$ is causal and has a *finite* duration L. This hypothesis is actually quite realistic. The only problem is that the complexity of the processes increases with L. Remember that we have already discussed the case where $k \neq 0$: this is the case where the ISI is equal to 0.

In the end, we can describe the digital transmission line as a "black box", with the sequence of symbols $a(n)$ ($\in \{-1, 1\}$) as the input, and as the output the sequence of real values given by the expression:

$$x_a(n) = g_0 a(n) + g_1 a(n-1) + \cdots + g_{L-1} a(n - L + 1) + b(n) \tag{12.65}$$

The choice of the likeliest decision sequence is based on the observations $x_a(n)$ for $n \in \{1, \ldots, N\}$. In the example we are going to discuss, we have chosen $L = 3$.

In practice, the emission and reception filters are known, whereas the channel filter is not. We have already said that in order to measure the quantities $g(k)$, we could then use a set sequence of symbols called the *training sequence.*

From now on, we will assume that the values of g_0, g_1, …, g_{L-1} are known. If we then use the Gaussian distribution of the sequence $b(n)$ and expression 12.65, the probability density of the observed sequence $\{x_a(1), \ldots, x_a(N)\}$ can be written:

$$p_X(\mathbf{x}; a_1, \ldots, a_N) = \frac{1}{(\sigma_b\sqrt{2\pi})^N} \exp\left(-\frac{1}{2\sigma_b^2} A\right) \tag{12.66}$$

$$\text{with } A = \sum_{n=1}^{N} \left[x_a(n) - g_0 a(n) - g_1 a(n-1) - g_2 a(n-2)\right]^2$$

where we have assumed that $a(0) = a(-1) = -1$. Based on the N observations $\{x_a(1), \ldots, x_a(N)\}$, the *maximum likelihood rule* consists of finding the sequence $\{a(1), \ldots, a(N)\}$ that maximizes $p_X(\mathbf{x}; a_1, \ldots, a_N)$, that is to say the sequence that minimizes the quantity:

$$\ell(\{x_a\}, \{a\}) = \sum_{n=1}^{N} [x_a(n) - g_0 a(n) - g_1 a(n-1) - g_2 a(n-2)]^2 \quad (12.67)$$

Theoretically, there are 2^N possible sequences and we could just bluntly compute the 2^N values of $\ell(\{x_a\}, \{a\})$ for the observed sequence $\{x_a(1), \ldots, x_a(N)\}$, and choose the sequence a that leads to the minimum.

The Viterbi algorithm makes it possible to cut down to $N \times 2^L$ the number of values that have to be calculated. Before we present this algorithm, we are going to study a simpler, but suboptimal process.

12.17.1 Linear equalization

The first idea consists of using the filter with the impulse response $w(n)$ that inhibits the effect of the channel $\{g_0,\ g_1,\ g_2\}$. This is called equalizing the channel and the filter $w(n)$ is called an *equalizer.*

There are two common approaches:

- The equalizer is chosen so that, in the absence of noise, it completely eliminates the intersymbol interference. This is called *Zero Forcing.*
- The equalizer is chosen so as to minimize, in the presence of noise, the square deviation between the original signal and the signal after it has been equalized. This is called the Wiener equalization. It requires for the noise's variance σ_b^2 to be known. The expression of its complex gain is given by equation 11.53:

$$W(f) = \frac{G^*(f)}{|G(f)|^2 + \sigma_b^2/\sigma_x^2}$$

Obviously, in the absence of noise ($\sigma_b = 0$), the two solutions coincide.

Exercise 12.26 ("Zero Forcing" linear equalization)

In the absence of noise, the Zero Forcing equalizer eliminates the ISI. Its transfer function is $W(z) = 1/G(z)$ with $G(z) = g_0 + g_1 z^{-1} + g_2 z^{-2}$. If $G(z)$ has its zeros inside the unit circle, the solution is stable and causal. Otherwise, the stable solution has a non-causal impulse response. It is possible to implement it by approximating it with a finite delay (see exercise 12.24).

Because $G(z)$ is a polynomial, that is to say a filter with a finite impulse response, the series expansion of $W(z)$ has an infinite number of coefficients and hence the filter has an infinite impulse response. In practice, it is often approximated by a filter with a long enough finite impulse response. Here we are going to design the filter $W(z) = 1/G(z)$ in its exact form using the MATLAB® function `filter`, which is possible only if $G(z)$ has all its zeros strictly inside the unit circle.

1. Using equation 12.65, show that the output signal $y(n)$ of the equalizer $W(z)$ can be expressed as $y(n) = a(n) + u(n)$, where $u(n)$ is a noise.

2. Let $g_0 = 1$, $g_1 = -1.4$ and $g_2 = 0.8$. Notice that the zeros are inside the unit circle, and therefore the equalizer $w(n)$ is stable and causal. Determine the variance of $u(n)$. You can use expression 8.63.

3. Use the result to find the probability distribution of $y(n)$ when $a(n) = -1$ and when $a(n) = +1$.

4. Given the previous, we have come up with the following rule: if $y(n)$ is positive, then the decision is that $a(n) = 1$, and otherwise the decision is that $a(n) = -1$. Determine the expression of the error probability.

5. Write a program that:

   - generates a random binary sequence, made up of -1 and $+1$, with a length N;
   - filters the resulting sequence by the filter with the impulse response `gc=[1 -1.4 0.8]`;
   - adds a Gaussian noise with a variance σ^2 such that the signal-to-noise ratio is equal to R dB;
   - passes the obtained signal through a filter with the transfer function $W(z) = 1/G(z)$;
   - compares the equalizer's output to the threshold 0 to decide what symbol was transmitted;
   - evaluates the number of errors (bear in mind that for the evaluation to be relevant, about a hundred errors have to be counted, hence N must be chosen high enough);
   - compares the results with the theoretical plot.

Exercise 12.27 (Wiener equalization)
Consider again equation 12.65:

$$x(n) = g_0 a(n) + g_1 a(n-1) + \cdots + g(L-1)a(n-L+1) + b(n) \quad (12.68)$$

$a(n)$ is assumed to be a sequence of equally distributed i.i.d. random variables with possible values in $\{-1, +1\}$. This means that $\mathbb{E}\{a(n)\} = 0$ and that $\mathbb{E}\{a(n+k)a(n)\} = \delta(k)$.

We wish to find a filter with a finite impulse response $w(n)$, with a length N, that minimizes the square deviation:

$$\mathbb{E}\left\{ |a(n-d) - \hat{a}(n)|^2 \right\} \tag{12.69}$$

where $\hat{a}(n) = x(n) \star w(n)$ refers to this filter's output. This filter is an example of the Wiener filter. d is a positive integer that accounts for the fact that a delay is required, because if the filter $h(n)$ is not minimum phase, then we know that the stable inverse is not causal. In practice, a delay must therefore be introduced to obtain a proper causal approximation.

1. Determine, as a function of the autocovariance function $R_{aa}(k)$ of the sequence $a(n)$ and of the covariance function $R_{ax}(k)$ between $a(n)$ and $x(n)$, the filter $w(n)$ that minimizes 12.69.
2. Determine the expression of $R_{ax}(k)$ as a function of $R_{aa}(k)$, and the expressions of $g(k)$ and of the autocovariance function $R_{bb}(k)$ of the noise $b(n)$.
3. Write the problem in matrix form. Determine the solution's expression.
4. Write a program that performs the equalization.
5. Write a program that uses the equalized output and applies symbol-by-symbol decision. Compare the results, in terms of error probability, with those obtained with the Zero Forcing equalizer.

12.17.2 The Viterbi algorithm

As we said, based on the N observations, we have to calculate the 2^N quantities:

$$\ell(\{x_a\}, \{a\}) = \sum_{n=1}^{N} \left[x(n) - g_0 a(n) - g_1 a(n-1) - g_2 a(n-2)\right]^2$$

corresponding to the 2^N length N binary sequences. Let us assume that, initially, $a(0) = -1$ et $a(-1) = -1$. Let $s(n) = [a(n-1) \quad a(n-2)]^T$ be the size 2 vector constructed by concatenating 2 consecutive symbols. In our case, $s(n)$ can only assume 4 different values, denoted symbolically by {00, 01, 10, 11}.

Let us calculate the probabilities for $s(n+1)$ to be equal to 00, 01, 10 et 11, respectively, knowing that $s(n) = 00$. If $s(n) = 00$, the only possible states for $s(n+1)$ are 00 if $a(n) = -1$ and 10 if $a(n) = +1$. Therefore:

$$\begin{aligned}
&\Pr(s(n+1) = 00|s(n) = 00) = 1/2\\
&\Pr(s(n+1) = 01|s(n) = 00) = 0\\
&\Pr(s(n+1) = 10|s(n) = 00) = 1/2\\
&\Pr(s(n+1) = 11|s(n) = 00) = 0
\end{aligned}$$

The probabilities of $s(n+1)$ with respect to the three other values of $s(n)$ can be calculated in the same way and represented by Figure 12.53.

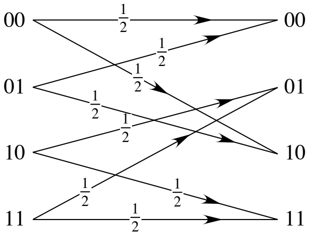


Figure 12.53 – *Transition probability graphs for the* 4 *states depending on the input symbol*

The probabilities can be grouped together in a *transition matrix*:

$$\begin{bmatrix} 1/2 & 0 & 1/2 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 1/2 & 0 & 1/2 \end{bmatrix}$$

representing the state transition probability for two consecutive times. This description of the evolution of $s(n)$ is called a *Markov*, or *Markovian* model. The Markovian property is what makes it possible to use the Viterbi algorithm. Consider again the expression we wish to minimize, and let $\mathrm{g} = [g_1 \quad g_2]^T$. We are going to use the definition of $s(n)$. If we stop the computation at step p, we get:

$$d(s(p),p) = \sum_{n=1}^{p} (x_a(n) - g_0 a(n) - \mathrm{g}^T s(n))^2$$

From now on, the quantity $d(s(p),p)$ will be called the *path metric* ($a(1)$, …, $a(p)$) at step p.

Let us assume that at step p, we still have 4 sequences (a_1, …, a_p) competing each other ending with $s(p)$ equal to 00, 01, 10 or 11 respectively and with the metric $d(00,p)$, $d(01,p)$, $d(10,p)$ and $d(11,p)$ respectively.

In order to calculate the metric of the sequence ($a(1)$, …, $a(p+1)$), we then have to the positive term $(x(p+1) - g_0 a(p+1) - \mathrm{g}^T s(p+1))^2$ for the two possible values of $a(p+1)$, that is to say -1 and $+1$. Hence each states has *two possible children*. We will therefore have to calculate, as functions of the observation $x_a(p+1)$ at the time $(p+1)$, 8 values that lead to one of the

4 possible states. These calculations are summed up in the following table:

$s(p)$		$s(p+1)$	
00	$\rightarrow$	00	$v(00,00) = d(00,p) + (x_a(p+1) - (-g_0 - g_1 - g_2))^2$
	$\searrow$	10	$v(00,10) = d(00,p) + (x_a(p+1) - (+g_0 - g_1 - g_2))^2$
01	$\rightarrow$	00	$v(01,00) = d(01,p) + (x_a(p+1) - (-g_0 - g_1 + g_2))^2$
	$\searrow$	10	$v(01,10) = d(01,p) + (x_a(p+1) - (+g_0 - g_1 + g_2))^2$
10	$\rightarrow$	01	$v(10,01) = d(10,p) + (x_a(p+1) - (-g_0 + g_1 - g_2))^2$
	$\searrow$	11	$v(10,11) = d(10,p) + (x_a(p+1) - (+g_0 + g_1 - g_2))^2$
11	$\rightarrow$	01	$v(11,01) = d(11,p) + (x_a(p+1) - (-g_0 + g_1 + g_2))^2$
	$\searrow$	11	$v(11,11) = d(11,p) + (x_a(p+1) - (+g_0 + g_1 + g_2))^2$

Notice that there are two ways of ending up in each state. For example we end up in $s(p+1) = 10$ either by starting in $s(p) = 00$, if we choose $a(p+1) = 1$, either by starting in $s(p) = 01$, if we choose $a(p+1) = 1$. These two possibilities correspond to the two values $v(00,10)$ and $v(01,10)$ respectively. It is useless to keep the largest one of the two, since any further value will have a higher metric. Hence $d(10, p+1) = \min(v(00,10), v(01,10))$.

In the end, the algorithm calculates, at step $p+1$:

$$\begin{aligned}
d(00,p+1) &= \min\{d(00,p) + [x_a(p+1) - (-g_0 - g_1 - g_2)]^2, \\
&\qquad d(01,p) + [x_a(p+1) - (-g_0 - g_1 + g_2)]^2\} \qquad (12.70)\\
d(01,p+1) &= \min\{d(10,p) + [x_a(p+1) - (-g_0 + g_1 - g_2)]^2, \\
&\qquad d(11,p) + [x_a(p+1) - (-g_0 + g_1 + g_2)]^2\} \qquad (12.71)\\
d(10,p+1) &= \min\{d(00,p) + [x_a(p+1) - (+g_0 - g_1 - g_2)]^2, \\
&\qquad d(01,p) + [x_a(p+1) - (+g_0 - g_1 + g_2)]^2\} \qquad (12.72)\\
d(11,p+1) &= \min\{d(10,p) + [x_a(p+1) - (+g_0 + g_1 - g_2)]^2, \\
&\qquad d(11,p) + [x_a(p+1) - (+g_0 + g_1 + g_2)]^2\} \qquad (12.73)
\end{aligned}$$

At each step, all the *parents* leading to the smallest metric have to be memorized. For example if the minimum of $d(00, p+1)$ is obtained for $d(01,p) + (x_a(p+1) - (-g_0 - g_1 + g_2)^2)$, it will be memorized as the state 00 at the time p that leads to the state 01 at the time $p+1$. The same is done for the three other states. The initial values are calculated based on the fact that we have assumed $a(0) = a(-1) = -1$, that is to say $s(0) = 00$. This means that when the first symbol is transmitted, either the state 00 or the state 10 is reached, and therefore:

$$\begin{aligned}
d(00,1) &= [x_a(1) - (-g_0 - g_1 - g_2)]^2\\
d(10,1) &= [x_a(1) - (hg_0 - g_1 - g_2)]^2
\end{aligned}$$

We then infer that:

$$\begin{aligned}
d(00,2) &= d(00,1)+[x_a(2)-(-g_0-g_1-g_2)]^2\\
d(01,2) &= d(10,1)+[x_a(2)-(-g_0+g_1-g_2)]^2\\
d(10,2) &= d(00,1)+[x_a(2)-(+g_0-g_1-g_2)]^2\\
d(11,2) &= d(10,1)+[x_a(2)-(+g_0+g_1-g_2)]^2
\end{aligned}$$

Obviously, the intermediate metrics do not need to be memorized. Only the four current ones have to be. Hence, formulae 12.70 to 12.73 have to be used as updating formulae for the 4 states reached at the considered state. The algorithm stops at the end of the length N observation sequence. Then the sequence considered to be the most likely is the one that leads to the state with the smallest metric. The sequence of states is determined by going back up the table of parents corresponding to this sequence. Once the sequence of states has been obtained, we simply compute the symbol sequence. The **viterbi.m** program determines the emitted sequence using the Viterbi algorithm. The results are given in Figure 12.54. The plot ('o') represents the error probability when using the Viterbi algorithm. The plot ('×') reproduces the results provided by the program from exercise 12.24 (a Zero Forcing equalization followed by a symbol-by-symbol threshold detection) by typing, after running it:

```
hold on; semilog(RSBdB, PeTheo,'x'); hold off
```

As you can see, the Viterbi results are much better than the ones obtained by linear equalization.

```
%===== VITERBI.M
N=5000; hc=[1 -1.4 0.8]; RSBdB=(5:17); longRSB=length(RSBdB);
PeVi=zeros(longRSB,1); ak=sign(randn(1,N));
ak(1)=-1; ak(2)=-1; sk=filter(hc,1,ak); vs=sqrt(sk*sk'/N);
%=====
dec=NaN*ones(4,4); dec(1,1)=-1; dec(2,1)=1; dec(3,2)=-1;
dec(4,2)=1; dec(1,3)=-1; dec(2,3)=1; dec(3,4)=-1; dec(4,4)=1;
%=====
for jj=1:longRSB
    RSB=10^(RSBdB(jj)/20); sigma_b=vs/RSB;
    bk=sigma_b*randn(1,N); xk=sk+bk;
    %==== Indexing 4 states (with the metric d2)
    %     1 for 00, 2 for 01, 3 for 10, 4 for 11
    %==== asc is the parents' sequence
    d2=zeros(4,1); asc=zeros(4,N);
    %==== Initialization
    d21=(xk(3)-(-hc(1)-hc(2)-hc(3)))^2;
    d23=(xk(3)-(hc(1)-hc(2)-hc(3)))^2;
    asc(1,3)=1; asc(3,3)=1;
    %====
    d2(1)=d21+(xk(4)-(-hc(1)-hc(2)-hc(3)))^2;
```

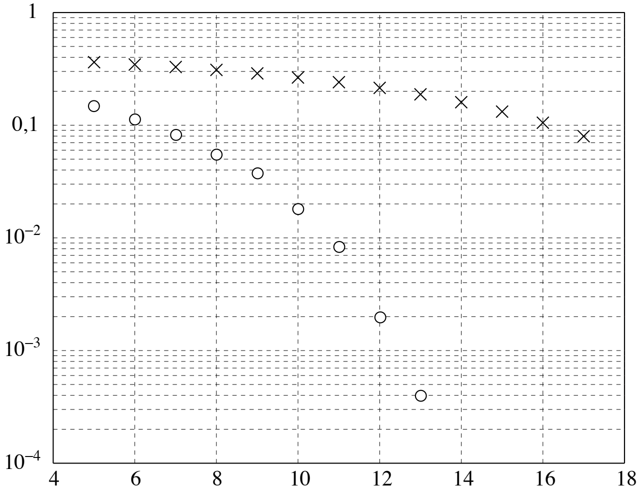


Figure 12.54 – *Comparing the error probabilities as functions of the signal-to-noise ratio in dB. Plot ('×'): zero forcing linear equalization. Plot ('o'): Viterbi algorithm. Results obtained through a simulation with a length of* 5,000*. The channel filter has the finite impulse response* $(1\ \ -1.4\ \ 0.8)$

```
    d2(2)=d23+(xk(4)-(-hc(1)+hc(2)-hc(3)))^2;
    d2(3)=d21+(xk(4)-(hc(1)-hc(2)-hc(3)))^2;
    d2(4)=d23+(xk(4)-(hc(1)+hc(2)-hc(3)))^2;
    asc(:,4)=[1;3;1;3]; d2kk=zeros(4,1); ind=zeros(4,1);
    for kk=5:N   %======
  [d2kk(1) ind(1)]=min([d2(1)+(xk(kk)-(-hc(1)-hc(2)-hc(3)))^2;...
         d2(2)+(xk(kk)-(-hc(1)-hc(2)+hc(3)))^2]);
  [d2kk(2) ind(2)]=min([d2(3)+(xk(kk)-(-hc(1)+hc(2)-hc(3)))^2;...
         d2(4)+(xk(kk)-(-hc(1)+hc(2)+hc(3)))^2]);
  [d2kk(3) ind(3)]=min([d2(1)+(xk(kk)-(hc(1)-hc(2)-hc(3)))^2;...
         d2(2)+(xk(kk)-(hc(1)-hc(2)+hc(3)))^2]);
  [d2kk(4) ind(4)]=min([d2(3)+(xk(kk)-(hc(1)+hc(2)-hc(3)))^2;...
         d2(4)+(xk(kk)-(hc(1)+hc(2)+hc(3)))^2]);
  asc(:,kk)=[1*(ind(1)==1)+2*(ind(1)==2);...
  3*(ind(2)==1)+4*(ind(2)==2);...
     1*(ind(3)==1)+2*(ind(3)==2);3*(ind(4)==1)+4*(ind(4)==2)];
        d2=d2kk;
    end          %======
    [metN indN]=min(d2); akVi=zeros(1,N-2); akVi(1)=-1; akVi(2)=-1;
    for kk=N:-1:3
        EI=asc(indN,kk); EF=indN; akVi(kk)=dec(EI,EF); indN=EI;
    end
    nbe=sum(abs(ak(1:N-2)-akVi(3:N))/2); PeVi(jj)=nbe/N;
end
semilogy(RSBdB, PeVi,'o'); grid
```

Part III

Hints and Solutions

This page intentionally left blank

Chapter 13

Hints and Solutions

H1 Signal fundamentals

H2 Discrete time signals and sampling

H2.1 (An illustration of the sampling theorem) (see page 63)

1. Because $F_s = 500$ Hz is greater than twice the signal's frequency (that is, 2×200 Hz), the sampling makes it possible to perfectly reconstruct the signal. Hence we end up with the same sine at the 200 Hz frequency.

2. Because $F_s = 250$ Hz is smaller than twice the signal's frequency, the sampling introduces aliasing. The $\pm F_s$ shifts in the spectrum (corresponding to $n = \pm 1$ in formula 2.5) contribute to the frequency with $-250 + 200 = 50$ Hz. Since the spectrum is symmetrical, everything happens as if the 200 Hz frequency were "aliased" by symmetry about the frequency $F_s/2 = 125$ Hz. The result of the reconstruction is a sine with the frequency 50 Hz (Figure H2.1).

3. Type:

```
%===== CECHAN2.M
Ds=.1;                % Signal length
F0=200;               % Frequency of the sine function
Fs=input('Sampling frequency in Hz (F0=200 Hz) = ');
Ts=1/Fs; Ne=Ds/Ts+1; % Number of samples
K=40;                 % Interpolation fonction for displaying
Tc=Ts/K; Nc=Ds/Tc+1; % Nb points of the "continuous" signal
%=================
tpc=[0:Nc-1]*Tc; xtc=cos(2*pi*tpc*F0); % "Continuous" signal
tpe=[0:Ne-1]*Ts; xte=cos(2*pi*tpe*F0); % Samples
subplot(211); plot(tpc,xtc,'-',tpe,xte,'o');
%===== Interpolation function
```

```
ht=sin(pi*Fs*tpc) ./ tpc /Fs / pi; ht(1)=1;
Ni=200;                          % Reconstruction filter
hti=[ht(Ni:-1:2) ht(1:Ni)];    % (length 2*Ni-1)
subplot(212); plot([-Ni+1:Ni-1]*Tc,hti); grid
%===== Reconstructed signal
xtr=zeros(1,Nc); xtr(1:K:Nc)=xte;
xti=filter(hti,1,[xtr zeros(1,Ni-1)]); Lxti=length(xti);
xti=xti(Ni:Lxti);     % Delay of the filter
subplot(211); hold on; plot(tpc,xti,'-r'); hold off
grid; set(gca,'xLim',[.03 .06]);  % Zoom in
```

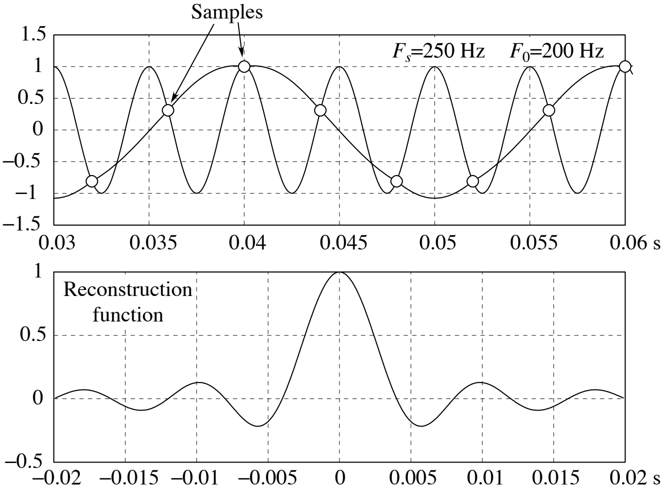


Figure H2.1 – *Sampling and reconstruction*

H2.2 (Time domain hermitian symmetry) (see page 71)

1. If we take the conjugate complex of $X(f)$ and use $x(n) = x^*(-n)$:

$$\begin{aligned} X^*(f) &= \sum_{n=-\infty}^{+\infty} x^*(n)e^{2j\pi nf} = \sum_{n=-\infty}^{+\infty} x(-n)e^{2j\pi nf} \\ &= \sum_{n=-\infty}^{+\infty} x(n)e^{-2j\pi nf} = X(f) \end{aligned}$$

If, furthermore, $x(n)$ is real, then we know that $X(f) = X^*(-f)$, hence $X(f) = X(-f) = X^*(f)$. The conclusion is that $X(f)$ is real and even.

2. The DTFT of $\{y(n)\}$ has the expression $Y(f) = \sum_{n=1}^{+\infty} x(n)e^{-2j\pi nf} + x(0)/2$. If we take the conjugate and use the fact that $x^*(n) = x(-n)$, then we have:

$$\begin{aligned} Y^*(f) &= \sum_{n=1}^{+\infty} x^*(n)e^{2j\pi nf} + x^*(0)/2 = \sum_{n=1}^{+\infty} x(-n)e^{2j\pi nf} + x(0)/2 \\ &= \sum_{k=-1}^{-\infty} x(k)e^{-2j\pi kf} + x(0)/2 \end{aligned}$$

Therefore, $Y^*(f) + Y(f) = 2\text{Re}\{Y(f)\} = X(f)$. This leads us to the following method for calculating the DTFT of a sequence $x(n)$:

(a) only the elements of $x(n)$ with non-negative indices are considered $(n \geq 0)$;

(b) the value $x(0)$ is divided by 2;

(c) the DTFT of the resulting sequence is calculated;

(d) the DTFT of the real part is calculated, then multiplied by 2.

H2.3 (Comparing computation speeds) (see page 73)
Type:

```
%===== COMPARE.M
clear; x=randn(1,1024); P=100;
tbcd=[]; tbcf=[]; % Table of the durations
for k=7:10,
    npts=2^k;     % Number of frequency points
    freq=[0:npts-1]/npts; n=[0:npts-1]';
    y=x(1:npts);  % Same number of samples
    t0 = clock;   % Direct computation =====================
    for m=1:npts
        fr=freq(m); caldir(m)=y * exp(-2*pi*j*fr*n);
    end
    te=etime(clock,t0); tbcd=[tbcd te];
    t0 = clock;             % Computing with the FFT =====
    for m=1:P               % We repeat it to get
        calfft=fft(y,npts); % a significant duration
    end
    te=etime(clock,t0); tbcf=[tbcf te];
end
format long; [tbcd' tbcf'/P (tbcd'./tbcf')*P]; format short
```

H2.4 (Spectrum of the triangle function) (see page 74)
Type:

```
%===== CSPECTRI.M
nfft=512; freq=[0:nfft-1]/nfft;
sig=[1:10 9:-1:0]; plot([0:19],sig,[0:19],sig,'x')
sigspec=fft(sig,nfft);
subplot(411); plot(freq,abs(sigspec)); grid;
subplot(412); plot(freq,angle(sigspec)); grid;
subplot(223); plot(freq,real(sigspec)); grid;
subplot(224); plot(freq,imag(sigspec)); grid;
```

We have to check, using Figure H2.2, that the DTFT obeys the hermitian symmetry property. Because of the periodicity with period 1, it means that the graphs are symmetrical about the frequency $f = 1/2$. The modulus and the real part are even, whereas the phase and the imaginary part are odd. The `unwrap` function can be used to plot the phase.

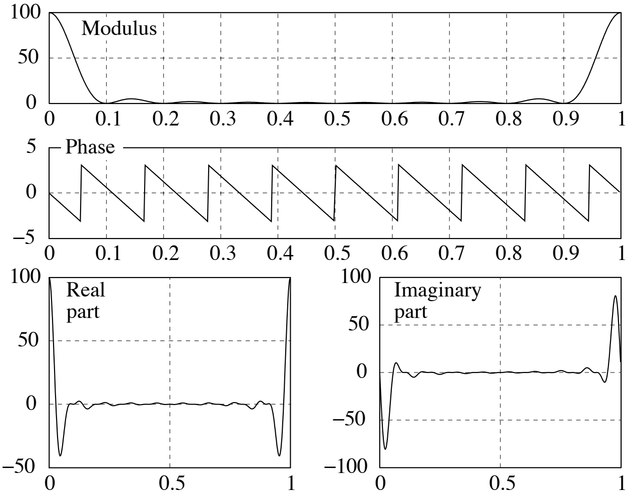


Figure H2.2 – *Spectral features of the triangle function. Above, the even modulus and the odd phase. Below, the even real part and the odd imaginary part*

H2.5 (Circular convolution of the rectangular signal) (see page 75)
The difference comes from the fact that, in the second case, the sequence **x** is padded with 8 zeros and the resulting convolution is linear.

H2.6 (Delay) (see page 75)

1. Let us calculate the DTFT over L points, that is to say the DFT. We

get:

$$\begin{aligned} Y(k/L) &= \sum_{n=0}^{n_1} x(n)e^{-2j\pi nk/L} + \sum_{n=L-n_0}^{L_1} x(n-L)e^{-2j\pi nk/L} \\ &= \sum_{n=0}^{n_1} x(n)e^{-2j\pi nk/L} + \sum_{p=-n_0}^{-1} x(p)e^{-2j\pi(p+L)k/L} = X(k/L) \end{aligned}$$

where we have defined $p = n - L$ and where we have used the property $e^{-2j\pi k} = 1$. To obtain the DTFT over L points of a signal that assumes non-zero values between the indices $-n_0$ and n_1, we have to shift by L the negative index values and calculate the DFT of the resulting sequence.

2. Type the program:

```
%===== CDECAL1.M
Lfft=256; fq=(0:Lfft-1)/Lfft;
n0=5; n1=5; xt=ones(n0+n1+1,1);
zs=zeros(Lfft-n0-n1-1,1);
yt=[xt(n0+1:n0+n1+1);zs;xt(1:n0)];
xf=fft(yt,Lfft); plot(fq,real(xf)); pause
plot(fq,imag(xf)) ; % Almost zero
```

H2.7 (FFTs of real sequences) (see page 79)

1. Let $A(k)$ be the DFT of $x(2n)$. Because $x(2n) = (y(n) + y^\star(n))/2$ and because the DFT of $y^*(n)$ is equal to $Y^*(-k \bmod N)$, we have:

$$A(k) = \frac{1}{2}\left(Y(k) + Y^*(-k \bmod N)\right) \tag{13.1}$$

Likewise, if $B(k)$ refers to the DFT of $x(2n+1)$, and because $x(2n+1) = (y(n) - y^\star(n))/2j$, we have:

$$B(k) = -\frac{j}{2}\left(Y(k) - Y^*(-k \bmod N)\right) \tag{13.2}$$

The two relations 13.1 and 13.2 allow us to directly calculate $A(k)$ and $B(k)$ from $Y(k)$.We will now see how $X(k)$ is obtained from $A(k)$ and $B(k)$.

2. Because $A(k) = \sum_{n=0}^{N/2-1} x(2n)W_{N/2}^{nk}$ et $B(k) = \sum_{n=0}^{N/2-1} x(2n+1)W_{N/2}^{nk}$, we get for $k \in \{0, \ldots, N-1\}$:

$$\begin{aligned} X(k) &= \left(\sum_{n=0}^{N/2-1} x(2n)W_N^{2nk}\right) + W_N^k \left(\sum_{n=0}^{N/2-1} x(2n+1)W_N^{2nk}\right) \\ &= A(k \bmod N/2) + W_N^k B(k \bmod N/2) \end{aligned}$$

This last part can be written as follows:

$$\left\{\begin{array}{ccl} A_0 & + & B_0 \\ A_1 & + & W_N^1 B_1 \\ \vdots & \vdots & \vdots \\ A_{N/2-1} & + & W_N^{N/2-1} B_{N/2-1} \\ A_0 & + & W_N^{N/2} B_0 \\ \vdots & \vdots & \vdots \\ A_{N/2-1} & + & W_N^{N-1} B_{N/2-1} \end{array}\right. \Rightarrow \left\{\begin{array}{ccl} A_0 & + & B_0 \\ A_1 & + & W_N^1 B_1 \\ \vdots & \vdots & \vdots \\ A_{N/2-1} & + & W_N^{N/2-1} B_{N/2-1} \\ A_0 & - & B_0 \\ A_1 & - & W_N^1 B_1 \\ \vdots & \vdots & \vdots \\ A_{N/2-1} & - & W_N^{N/2-1} B_{N/2-1} \end{array}\right.$$

This tells us how to calculate the DFT of the real, length N sequence $x(n)$ using the DFT of the complex, length $N/2$ sequence $y(n)$:

(a) We define the sequence $y(n) = x(2n) + jx(2n+1)$ and calculate its $N/2$ order FFT $Y(k)$.

(b) We calculate $A(k)$ and $B(k)$ using relations 13.1 and 13.2 respectively.

(c) We calculate $X(k) = A(k) + W_N^k B(k)$.

3. The previous algorithm is comprised of $(N/2)\log_2(N/2)$ operations for the computation of the complex, length $N/2$ FFT, and also of N multiplication-additions for the computation of $X(k)$. As a consequence, the total computation is roughly $N/2\log_2(N/2) + N$ operations. This number should be compared with the computation load involved when using the length N algorithm, which is $N\log_2(N)$. For $N = 1{,}024$ we get 5,632 in the first case, whereas we get 10,240 in the second.

4. Simulation program (Figure H2.3):

```
%===== CFFTREEL.M
N=64; mtime=[0:N-1]; f0=.23; freq=[0:N-1]/N;
x=sin(2*pi*f0*mtime); x=[x zeros(1,rem(N,2))];
Nx=size(x,2); xspec=fft(x); xspeca=abs(xspec);
%===== Approximating the DTFT
atftd=fft(x,1024);
plot([0:1023]/1024,abs(atftd),’r’); grid; hold on
%===== Result of the direct calculation
plot(freq,xspeca);
set(gca,’xlim’,[0 .5],’ylim’,[0 max(abs(atftd))])
x2n=x(1:2:Nx-1); x2np1=x(2:2:Nx); Ny=Nx/2;
y = x2n + j*x2np1; yspec=fft(y); %
inds=[1 Ny:-1:2];                  % Conjugation
yspecs(1,:)=conj(yspec(inds));     % in time
```

```
Ak = (yspec+yspecs)/2; Bk = j*(yspecs-yspec)/2;
Wn=exp(-2*j*pi*[0:Nx/2-1]/Nx); Wn=[Wn -Wn];
yk = [Ak Ak] + [Bk Bk] .* Wn;
%===== Result
plot(freq,abs(yk),'or'); hold off
```

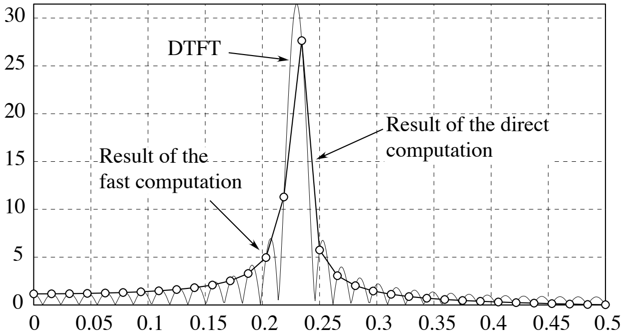


Figure H2.3 – *Comparing the results*

H2.8 (Using the FFT) (see page 80)

The program draws a unit circle, since the calculation of the FFT of [0 1] leads to the values:

$$X(k) = e^{-2\pi jk/128}, \text{ with } k = 0 \ldots 127$$

H3 Spectral observation

H3.1 (Study of the resolution) (see page 86)

1. The expression of $X(f)$ is:

$$X(f) = a_0 g_N(f - f_0) + a_1 g_N(f - f_1) + a_0 g_N(f + f_0) + a_1 g_N(f + f_1)$$

2. Let us assume that $f_0 \gg 1/N$, $f_1 \gg 1/N$ and $|f_0 - f_1| \gg 1/N$. Because $g_N(f)$ is a quickly decreasing function for $|f| \gg 1/N$, the four terms involved in the expression of $X(f)$ are never simultaneously null. Hence, for f belonging to a neighborhood of f_0:

$$|X(f)| \simeq a_0 |g_N(f - f_0)|$$

$|X(f)|$ shows a maximum in f_0, the amplitude of which is roughly Na_0.

3. Type the program:

```
%===== CRESOL.M
N=32;            % Signal length
L=1024;          % Number of frequency points
f0 = 0.2;
mtime=(0:N-1)';  % Column vector for time
freq=(0:L-1)/L;
phi=input('Relative phase between the two sines (degrees): ');
phi=phi*pi/180;
adb= input('Amplitude ratio (dB): ');
a = 10 ^(adb/20);
deltaf=(1/N:1/(5*N):3/N) ; % Frequency deviations
nbdf=length(deltaf); f1=f0+deltaf;
x1=cos(2*pi*mtime*f0); x1f=20*log10(abs(fft(x1,L)));
%===== For each f1
for k=1:nbdf
    x2=a * cos(2*pi*mtime*f1(k)+phi);
    x2f=20*log10(abs(fft(x2,L)));
    subplot(211); plot(freq,[x1f x2f]); grid
    axis([0 0.5 -20  40]);
    x=x1+x2; xf=20*log10(abs(fft(x,L)));
    subplot(212); plot(freq,xf); grid;
    axis([0 0.5 -20  40]);
    title(sprintf('delta_f = %1.2g x 1/N',deltaf(k)*N))
    disp('Press a key to proceed.')
    pause
end
```

As you can see, the resolution is highly dependent on the relative phase between the two sines.

H3.2 (Effect of the Hamming windowing) (see page 88)

1. Calculation of the normalization coefficient c_h: let $X_h(f)$ be the spectrum of the windowed signal. We have:

$$X_h(f) = A \sum_{n=0}^{N-1} c_h w_h(n) \exp(2j\pi f_0 n) \exp(-2j\pi f n)$$

At point f_0, we have $X_h(f_0) = A \sum_{n=0}^{N-1} c_h w_h(n)$, a quantity we wish to have equal to A. Therefore, $c_h = 1/\sum_{n=0}^{N-1} w_h(n)$.

2. Type:

```
%===== CEFFHAM.M
N=32; L=1024; freq=(0:L-1)/L;
s=exp(2*i*pi*0.2*(0:N-1));
whamm = 0.54 - 0.46 * cos(2*pi*(0:N-1)/N);
```

```
cr = 1 / N; ch = 1 / sum(whamm);
%===== Windowing (rectangle and hamming windows)
sr=cr * s; sh = ch * s .* whamm;
srf=fft(sr,L); shf = fft(sh,L);
srfdb=20 * log10(abs(srf)); shfdb=20 * log10(abs(shf));
subplot(211); plot(freq,shfdb); grid
[xy1 xy2]=ginput(2);
sprintf('Ratio=%d dB',abs(xy1(2)-xy2(2)))
subplot(212); plot(freq,srfdb); grid
[xy1 xy2]=ginput(2);
sprintf('Ratio=%d dB',abs(xy1(2)-xy2(2)))
```

3. Figure H3.1 shows the DTFT of the Hamming window. The width of the main lobe is measured, as well as the attenuation between the first side lobe and the main lobe (look up how to use the `ginput` function):

window	width	attenuation
rectangular	$2/N$	-13 dB
Hamming	$4/N$	-40 dB

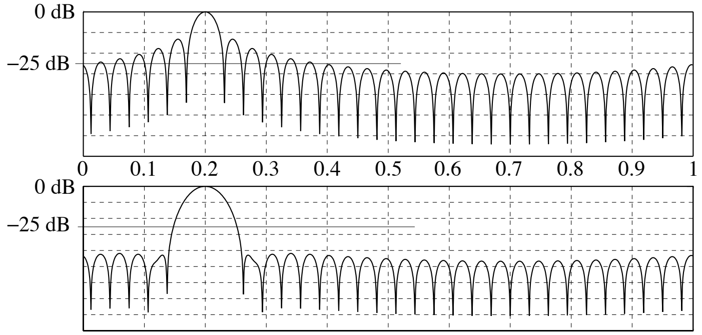


Figure H3.1 – *Comparing the rectangular window (above) and the Hamming window (below)*

4. In the case where the two sines have the same amplitudes, the resolution is directly related to the width of the main lobe, because if the two sines are distant enough from each other, then main lobes will not be too close, and they can easily be distinguish (see exercise 3.1).

   For $N = 32$ and $F_s = 1{,}000$ Hz, the resolutions obtained for 1.5 times the width of the main lobe are:

window	width	resolution(Hz)
rectangular	$1/N$	$1{,}500/32 \approx 47$ Hz
Hamming	$2/N$	$3{,}000/32 \approx 94$ Hz

5. In the case where the two sines have different amplitudes, the resolution is related to the width of the main lobe, but also to the height of the side lobe. For example, if we wish to distinguish a possible ratio of 25 dB, we have to go beyond the 6-th side lobe in the case of the rectangular window, whereas we only need to go beyond the first main lobe in the case of the Hamming window (see Figure H3.1). In this case, the Hamming window therefore allows a resolution of $1.5 \times 2{,}000/32 \approx 94$ Hz, which is better than the one obtained with a rectangular window, which is $6 \times 1{,}000/32 \approx 188$ Hz.

H3.3 (Short term Fourier transform) (see page 93)

1. The analysis function is as follows:

```
function [spec,normtm]=tfct(xt,Lb,ovlp,Lfft,win)
%%===================================================%
%% Short Term Fourier Transform                      %
%% SYNOPSIS: [spec,normtm]=TFCT(xt,Lb,ovlp,Lfft,win) %
%%      xt     = Signal                              %
%%      Lb     = Block size                          %
%%      ovlp   = Overlap length                      %
%%      Lfft   = FFT length                          %
%%      win    = window type                         %
%%      spec   = spectrogram                         %
%%      normtm = time vector (normalized)            %
%%===================================================%
if nargin<4, win='rect'; end
xt=xt(:); x=xt; Nx=length(xt);
if win=='hamm',
   wn=.54-.46*cos(2*pi*[0:Lb-1]'/Lb);
elseif win=='hann'
   wn=.5-.5*cos(2*pi*[0:Lb-1]'/Lb);
else
   wn=ones(Lb,1);
end
blkS=(Lb-ovlp); nbfen=floor(Nx/blkS); Lxb=nbfen*blkS;
%===== Calculating the index
idxH=[1:Lxb]; idxtab=reshape(idxH,blkS,nbfen);
indx=idxtab(blkS,:)+1; idxv=[1:ovlp-1]'*ones(1,nbfen);
idxh=ones(ovlp-1,1)*indx; idxtab2=[indx;idxv+idxh];
idxtab=[idxtab;idxtab2]; idxmax=idxtab(Lb,nbfen);
idlm=find(idxtab(Lb,:)>=Nx); nbf=idlm(1);
xx=zeros(Lb,nbf); x=[x;zeros(idxmax-Nx,1)];
xx(:)=x(idxtab(:,1:nbf)); Nc=size(xx,2);
xxp=xx .*(wn*ones(1,Nc));
spec=fft(xxp,Lfft); normtm=[0:Nc-1]*blkS;
return
```

2. The following program returns Figure H3.2 for 50 sample blocks:

```
%===== CTFDCT.M
%     Uses GENE1.M or GENE2.M
Tt=length(xt); Lfft=128; frq=Fs * (0:Lfft-1)/Lfft;
frqs2=frq(Lfft/2:-1:1);
disp(sprintf('Number of samples: %.0f',Tt));
tbl=input('Block size (0 to return)=');
win='rect';
while (tbl ~= 0)
    [spec,tps]=tfct(xt,tbl,floor(tbl/2),Lfft,win);
    mtime=tps/Fs;
    xreshf=abs(spec); xreshf=xreshf(Lfft/2:-1:1,:);
    contour(mtime,frqs2,xreshf); grid
    tbl=input('Block size (0 to return)=');
end
```

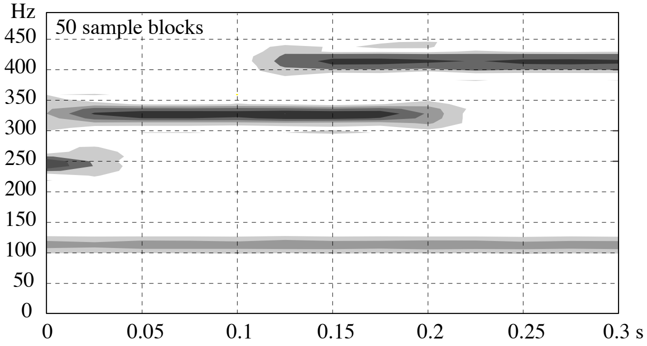


Figure H3.2 – *Short term study for* 50 *sample blocks*

H3.4 (Visualizing the aliasing with the STFT) (see page 94)

```
%===== MODULFREQ2.M
lambda=2000; Fs=8000; F0=1000; T=2;
nfft=128; Lbloc=100; freq=[0:nfft/2-1]*Fs/nfft;
%===== Signal
it=(0:Fs*T-1)/Fs;
theta=2*pi*F0*it+pi*lambda*(it .^2);
x=cos(theta'); Lx=length(x);
nblocs=floor(Lx/Lbloc); x=x(1:nblocs*Lbloc);
x=reshape(x,Lbloc,nblocs);
%===== Windowing
w=.54-.46*cos(2*pi*[0:Lbloc-1]'/(Lbloc-1));
w=w * ones(1,nblocs);
```

```
B=x.*w; A=abs(fft(B,nfft));
%===== Displaying between 0 and Fs/2
mesh([1:nblocs],freq,A(1:nfft/2,:))
view([-20 40])
```

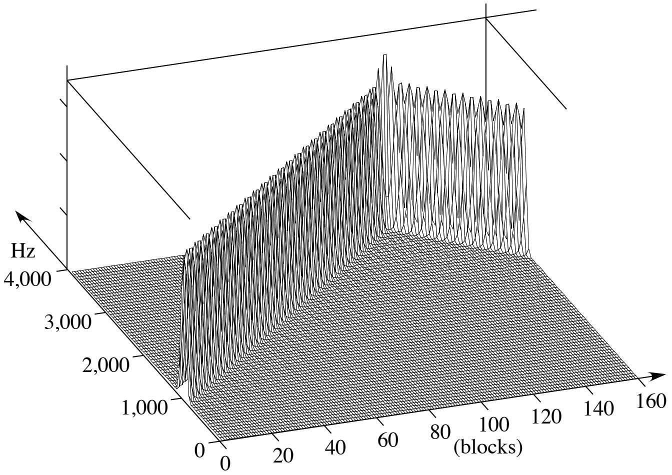


Figure H3.3 – *Time-frequency representation*

H3.5 (Effects of sampling and windowing) (see page 94)

1. The signal $x(t)$ has the following Fourier transform:

$$X(F) = \int_0^{+\infty} e^{-t/t_0} e^{-2j\pi F t} dt = \frac{t_0}{1 + 2j\pi F t_0}$$

Notice that $|X(F)|^2$ goes from the value t_0^2 in $F = 0$ to the value $t_0^2/2$ in $F_c = 1/(2\pi t_0)$. The frequency F_c which corresponds to a ratio of 2, hence $10\log_{10}(2) = 3$ dB, is called the 3 dB *cut-off frequency.*

2. $X_s(f)$ is obtained by periodizing F_s then normalizing the frequency scale by dividing by F_s. The fact that $X(F)$ has an infinite band causes aliasing. However this aliasing decreases as $F_s \gg F_c$.

3. This is equivalent to mutiplying the signal $x_s(n)$ by a width M rectangular window, hence to convolute $X_s(f)$ with the function given by expression 3.1. This results in oscillations with pseudo-period $1/M$.

4. Type:

```
%===== CEFFSAMP.M
clear; clf
%===== Continuous time
t0=1/0.7; Fa=100; M=10;
%===== Discrete time
Fs=2; Ts=1/Fs; Ntrq=5;
%===== Continuous time signal
tpsa=(0:M*Fa-1)/Fa; xa=exp(-tpsa/t0);
subplot(2,2,1); plot(tpsa,xa);
%===== FT
subplot(2,2,2);
frqsa=(-100:0.1:100)/Fa;
XFa= abs(t0 ./ (1+2*j*pi*t0*frqsa));
plot(frqsa,XFa,'-',frqsa+Fs,XFa,'g:',frqsa-Fs,XFa,'g:');
%===== Discrete time signal
subplot(2,2,1);
xe=xa(1:Ts*Fa:M*Fa); tpse=tpsa(1:Ts*Fa:M*Fa);
hold on; stem(tpse,xe); hold off
%===== DTFT
subplot(2,2,2);
Lfft=1024; frqstrq=Fs*(0:Lfft-1)/Lfft-Fs/2;
XFs=abs(fft(xe,Lfft))/Fs; XFs=fftshift(XFs);
hold on;
plot(frqstrq,XFs,'r',frqstrq+Fs,XFs,'r',frqstrq-Fs,XFs,'r');
plot([Fs Fs],[1.2 0],':'); plot([-Fs -Fs],[1.2 0],':')
hold off; set(gca,'xlim',[-3 3]);
%===== Truncated signal
subplot(2,2,3);
xatrq=xe(1:Ntrq); tpstrq=tpse(1:Ntrq);
stem(tpstrq,xatrq); set(gca,'xlim',[0 tpsa(M*Fa)])
%===== DTFT
frqstrq=Fs*(0:Lfft-1)/Lfft-Fs/2;
XFtrq=abs(fft(xatrq,Lfft))/Fs;
XFtrq=fftshift(XFtrq);
subplot(2,2,4);
plot(frqstrq,XFtrq,'r',frqstrq+Fs,XFtrq,'r',frqstrq-Fs,XFtrq,'r');
set(gca,'xlim',[-3 3]);
%==== TFD
LfftTFD=8; frqstrqTFD=Fs*(0:LfftTFD-1)/LfftTFD-Fs/2;
XFtrqTFD=abs(fft(xatrq,LfftTFD))/Fs;
XFtrqTFD=fftshift(XFtrqTFD);
hold on
stem(frqstrqTFD,XFtrqTFD);
plot([Fs Fs],[1.2 0],':'); plot([-Fs -Fs],[1.2 0],':')
hold off
```

H3.6 (Amplitude modulation) (see page 95)

1. We have $x(t) = \cos(2\pi F_0 t) + km(t)\cos(2\pi F_0 t)$. If we replace $\cos(2\pi F_0 t)$ with $[\exp(2j\pi F_0 t) + \exp(-2j\pi F_0 t)]/2$ and take the Fourier transform, we get:

$$\begin{aligned} 2X(F) &= \delta(F - F_0) + \delta(F + F_0) + kM(F) \star (\delta(F - F_0) + \delta(F + F_0)) \\ &= \delta(F - F_0) + \delta(F + F_0) + kM(F - F_0) + kM(F + F_0) \end{aligned}$$

   In any case, the spectrum of $x(t)$ contains two peaks at the frequencies $\pm F_0$, as well as the spectrum of $m(t)$ shifted by $\pm F_0$. If the width of $m(t)$ is B, meaning that its spectrum is non-zero between $-B$ and $+B$, then the spectrum of $x(t)$ occupies a $2B$ band around F_0. Because the spectrum of $m(t)$ obeys hermitian symmetry (real signal), the spectrum of $X(F)$ has the same property around F_0. Hence we can restrict the representation of $X(F)$ to the frequencies beyond F_0.

2. The spectrum is comprised of 7 peaks in the positive frequencies (the negative frequencies are obtained using hermitian symmetry):

   – 50 kHz (carrier);

   – 47,870 Hz and 52,130 Hz originating from the component at 2,130 Hz;

   – 46,250 Hz and 53,750 Hz originating from the component at 3,750 Hz;

   – 45,040 Hz and 54,960 Hz originating from the component at 4,960 Hz.

3. The program `cmodam.m` allows you to obtain Figure H3.4 for $m(t)$ and $x(t)$.

   The absence of overmodulation is characterized by the fact that $(1 + km(t))$ never becomes negative. Notice that $(1 + km(t))$ is therefore the the upper envelope of $x(t)$. This is an essential practical result, as it allows us to perform the demodulation operation in a very simple way: a full-wave rectifier is used, followed by an RC filter (see exercise 12.1) in order to detect the envelope. If $B \ll 1/RC \ll F_0$, the output signal will follow the envelope. The development of radiocommunications was based on this very simple technique.

4. Here, the two closest peaks are 1,000 Hz apart, or in normalized frequencies, 1,000/500,000 apart. Hence, in order to distinguish them using the DTFT, we need a number of points much greater than 500. To be able to have an outright separation we will choose $N = 1{,}000$, which corresponds to 2 ms of signal.

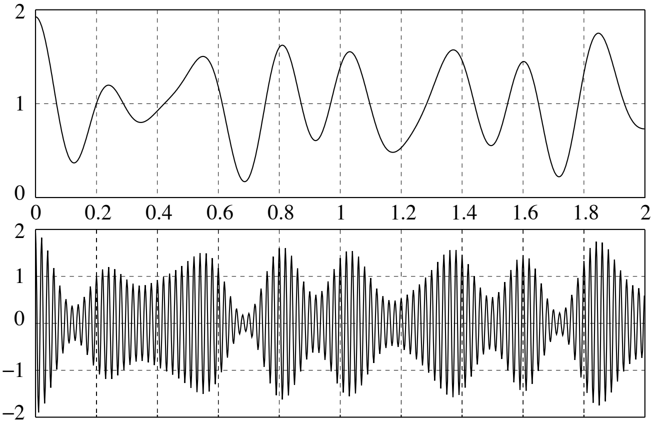


Figure H3.4 – *Amplitude Modulation. Above: signal $m(t)$. Below: modulated signal*

5. In normalized frequencies, 100 Hz correspond to 1/5,000. Hence we need an FFT size greater than 5,000.

6. Type the following program:

```
%===== CMODAM.M
Fs=500000; durat=2/1000; N=Fs*durat;
td=(0:N-1); t=td/Fs;        %
F0=50000; f0r=F0/Fs;        % Mod. frequency
Fm=[2130 ; 3750 ; 4960]; fmr=Fm/Fs; Am=[1 1.8 0.9];
k=1/2; ac=1+k*Am * cos(2*pi*fmr*td);
xt=ac .* cos(2*pi*f0r*td);
subplot(311); plot(t,ac); grid
subplot(312); plot(t,xt); grid
L=8192; freq=[0:L-1]/L*Fs; % Real frequency
subplot(313); plot(freq,abs(fft(xt,L))); grid
set(gca,'xLim',[40000 60000])
```

Figure H3.5 shows the spectrum obtained in agreement with the theoretical spectrum.

H3.7 (Carrierless double side-band) (see page 96)

1. If $M(F)$ refers to the Fourier transform of $m(t)$, then that of the modulated signal $x(t) = m(t)\cos(2\pi F_0 t)$ is $X(F) = (M(F + F_0) + M(F - F_0))/2$. The spectrum of $x(t)$ is comprised, around F_0, of the spectrum $M(F - F_0)/2$, which has a width of $2B$.

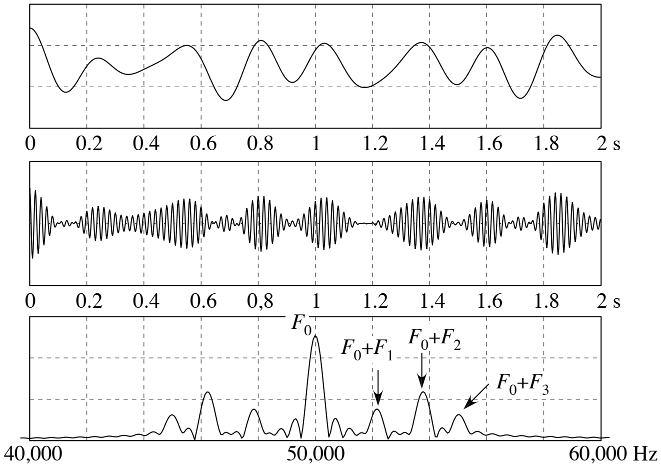


Figure H3.5 – *Spectrum of a double side band modulation*

2. If we multiply the signal $x(t)$ by $2\cos(2\pi F_0 t + \phi)$, we get $y(t) = 2m(t)\cos(2\pi F_0 t)\cos(2\pi F_0 t + \phi)$ which can also be written $y(t) = m(t)\cos(\phi) + m(t)\cos(4\pi F_0 t + \phi)$. The signal $y(t)$ therefore has a low frequency component $m(t)$, multiplied by $\cos(\phi)$, and a (modulation type) high frequency around $2F_0$ with a width $2B$. If we then use a low-pass filter, such as the one shown in Figure H3.6, the signal $m(t)\cos(\phi)$ is reconstructed.

   It is important to have $\phi = 0$, because if $\phi \neq 0$, the useful signal is attenuated. In the presence of noise, a mere amplification is not sufficient to compensate this attenuation.

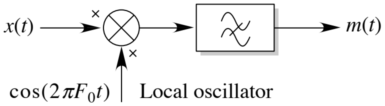


Figure H3.6 – *Synchronous demodulator*

3.

```
%===== CMODDBSP.M
Fs=500000; durat=2/1000; N=Fs*durat;
td=(0:N-1); t=td/Fs;        %
F0=50000; f0r=F0/Fs;        % Mod. frequency
Fm=[2130 ; 3750 ; 4960]; fmr=Fm/Fs; Am=[1 1.8 0.9];
mt=Am * cos(2*pi*fmr*td);
xt=mt .* cos(2*pi*f0r*td);
subplot(311); plot(t,mt); grid
subplot(312); plot(t,xt); grid
L=8192; freq=[0:L-1]/L*Fs; % Real frequency
subplot(313); plot(freq,abs(fft(xt,L))); grid
set(gca,'xLim',[40000 60000])
```

H3.8 (Stereophonic signal) (see page 97)

1. Spectrum of $c(t)$ (Figure H3.7):

$$\begin{aligned} C(F) &= (G(F)+D(F)) + \frac{1}{2}(G(F+F_0)+D(F+F_0)) \\ &\quad + \frac{1}{2}(G(F-F_0)+D(F-F_0)) + \frac{P}{2}\delta(F+\frac{F_0}{2}) + \frac{P}{2}\delta(f-\frac{f_0}{2}) \end{aligned}$$

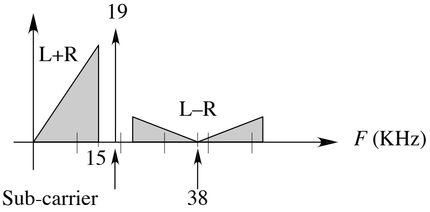


Figure H3.7 – *Spectrum of the stereophonic signal used for FM radio broadcasting*

2. For a monophonic set, all we need to do is filter the signal $c(t)$ in the $(-15,+15)$ kHz band to reconstruct the signal $g(t)+d(t)$. This is what determined the choice of the composite signal's shape. People who owned a monophonic set had to be able to still listen to it without having to buy a new set.

3. Type the program:

```
%===== CSTEREO.M
fa=1000000; f0=38000/fa;
fl=[380 957 1164 1587 1953]'/fa;
Al=[0.7 1.5 1.9 2.8 3.7];
fr=[347 523 1367 2465 3888]'/fa;
Ar=[0.3 1.5 2.7 1.7 2.3];
T=1000; t=(0:T-1);
%===== Left and right signals
g=Al*cos(2*pi*fr*t); d=Ar*sin(2*pi*fl*t);
c=(g+d)+(g-d) .* cos(2*pi*f0*t);
plot(t',[c' 2*g' 2*d']); grid
```

We need to sample the signal $c(t)$ at the frequency of 76 kHz. The odd times correspond to the left signal, and the even times to the right signal (Figure H3.8). Obviously, a slight delay causes crosstalk, meaning that a small part of the right signal is mixed up with the left signal, and vice versa.

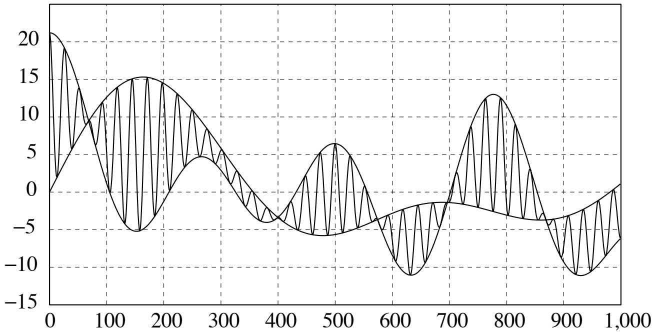


Figure H3.8 – *Composite stereophonic signal $c(t)$: the upper and lower envelopes represent the left and right signals*

H4 Linear filters

H4.1 (Rectangular impulse response filter) (see page 118)

1. This filter calculates the *mean* of the last M values of the input signal. This operation smooths the signal and eliminates the rapid fluctuations corresponding to the high frequencies. This is a *low-pass* filter with a cut-off frequency dependent on the value of M.

2. The complex gain is:

$$H(f) = e^{-j(M-1)\pi f}\frac{\sin(M\pi f)}{\sin(\pi f)}$$

and the phase is piecewise linear with the slope $-(M-1)\pi$:

$$\phi(f) = -(M-1)\pi f + \epsilon(f)\pi$$

where $\epsilon(f)$ is equal ± 1 depending on whether $\frac{\sin(M\pi f)}{\sin(\pi f)}$ is positive or negative.

3. These results are gathered in Figure H4.1 where the phase was represented using the `angle` function which brings it back between $-\pi$ and π.

Figure H4.1was obtained using the program:

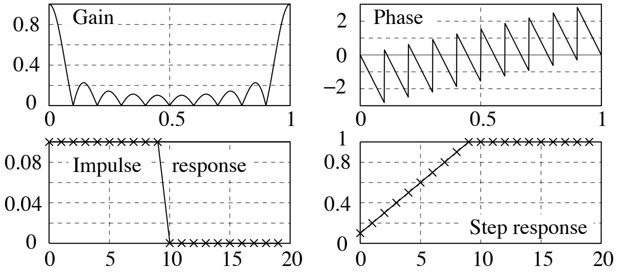


Figure H4.1 – *Gain, phase, impulse response and step response*

```
%===== RECTFILTER.M
T=20; mtime=(0:T-1); Lfft=1024; fq=(0:Lfft-1)/Lfft;
M=10; h=ones(1,M)/M;
Hf=fft(h,Lfft);                 % Frequency response
Gf=abs(Hf); Phif=angle(Hf);
d=eye(1,T); yd=filter(h,1,d);  % Impulse response
u=ones(1,T); yu=filter(h,1,u); % Step response
subplot(221); plot(fq,Gf); grid
subplot(222); plot(fq,Phif); grid
subplot(223); plot(mtime,yd,'-',mtime,yd,'x'); grid
subplot(224); plot(mtime,yu,'-',mtime,yu,'x'); grid
```

4. The filter resulting from cascading the two previous filters has as its impulse response the triangle function $(h \star h)(n)$. The corresponding gain is the square $H^2(f)$ of the previous gain.

H4.2 (Purely recursive first order) (see page 119)

1. Because the filter is causal, the convergence area is of the type $|z| > |a|$.

2. If we perform the series expansion of $H_z(z)$, and according to the definition of the z-transform, we have:

$$H_z(z) = 1 + az^{-1} + a^2z^{-2} + \cdots = \sum_{k\in\mathbb{N}} a^k z^{-k} \Rightarrow \begin{cases} h(n) = a^n \text{ for } n \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

The BIBO stability condition is satisfied since:

$$\sum_{k\in\mathbb{Z}} |h(k)| = \sum_{k\in\mathbb{N}} |a|^k < \infty \Rightarrow |a| < 1$$

3. The complex gain, the gain and the phase are:

$$H_z(e^{2\pi jf}) = H(f) = \frac{1}{1 - ae^{-2\pi jf}},\ \text{gain } = |H(f)|,\ \text{phase } = \arg(H(f))$$

4. We saw that the impulse response of the filter had the expression $h(n) = \lambda a^n$ for $n \geq 0$ and 0 otherwise. Therefore, the index response can be written:

$$y(n) = \lambda \sum_{k=0}^{n} a^k = \lambda \frac{1 - a^{n+1}}{1 - a}$$

If we choose $\lambda = 1 - a$, then $y(n) = 1 - a^{n+1}$ tends to 1 when n tends to infinity, with a decreasing speed as $|a|$ get closer to 1.

5. The plots H4.2 were obtained using the following program:

```
%===== REPINDICAR1.M
N=30; mtime=(0:N-1); a=[-2/3 1/2 3/4 7/8];
Na=length(a); indic=ones(N,1); y=zeros(N,Na);
for ii=1:Na
    y(:,ii)=filter(1-a(ii),[1 -a(ii)],indic);
end
plot(mtime,y,'-',mtime,y,'o');
set(gca,'xlim',[0 N-1]); set(gca,'ylim',[0 1.8]); grid
```

As you can see, as $|a|$ gets closer to 1, the output signal slowly converges to its limit value 1. This “rise time” can be evaluated from the index beyond which the difference with the value 1 is considered to be negligible. To be more precise, we can write that if $0 < a < 1$, $y(n) = 1 - e^{(n+1)\log(a)} = 1 - e^{-(n+1)/\tau}$ where $\tau = 1/\log(1/a) > 0$.

If $-1 < a < 0$, $y(n) = 1 - (-1)^{n+1} e^{-(n+1)/\tau}$ where $\tau = 1/\log(1/|a|) > 0$. Therefore, whether $a \in (-1, 1)$ is positive or negative, the index response has an “exponential” shape, the time constant of which is given by $\tau = 1/\log(1/|a|)$. The closer $|a|$ gets to 1, the larger τ becomes.

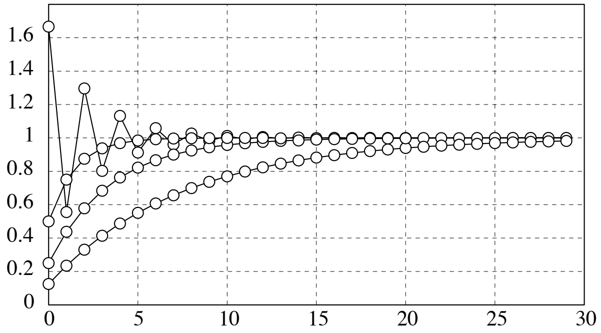


Figure H4.2 – *Step response of the filter* $H(z) = 1/1 - az^{-1}$, *for* $a = -2/3$, $a = 1/2$, $a = 3/4$ *and* $a = 7/8$

H4.3 (Purely recursive second order) (see page 123)

1. The transfer function is given by:

$$H_z(z) = \frac{1}{(1-p_1 z^{-1})(1-p_1^* z^{-1})} = \frac{1}{1-2\mathrm{Re}(p_1)z^{-1}+|p_1|^2 z^{-2}}$$

Therefore, $a_1 = -2\mathrm{Re}(p_1)$ and $a_2 = |p_1|^2$. The variations as functions of the phases of the poles is given by the program:

```
%===== CAR21.M
% Gain as a function of the phase with a constant modulus
Lfft=1024; freq=(0:Lfft-1)/Lfft; modp=0.9;
theta=(20:10:80); theta=theta * pi / 180;
nbph=length(theta);
a1=-2*modp*cos(theta); a2=modp ^2 * ones(1,nbph);
AA=[ones(1,nbph); a1; a2];
Df=fft(AA, Lfft); Hf=-20 * log10(abs(Df));
plot(freq(1:Lfft/2),Hf(1:Lfft/2,:)); grid
```

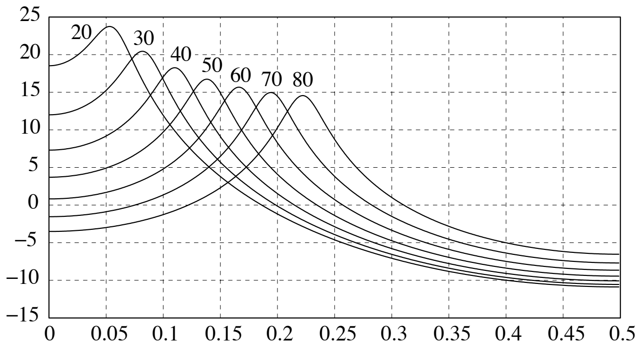


Figure H4.3 – *Gains of a second order filter as functions of the phases of the poles*

2. Type (Figure H4.4):

```
%===== CAR22.M
%==== Gain as a function of the modulus
Lfft=1024; freq=(0:Lfft-1)/Lfft;
modp=[0.1:0.2:0.9 .95 .98];     % A few moduli
theta=30 * pi / 180;
nbph=length(modp);
a1=- 2 * modp * cos(theta); a2=modp .^2;
AA=[ones(1,nbph); a1; a2];
Df=fft(AA, Lfft); Hf=-20 * log10(abs(Df));
plot(freq(1:Lfft/2),Hf(1:Lfft/2,:)); grid
```

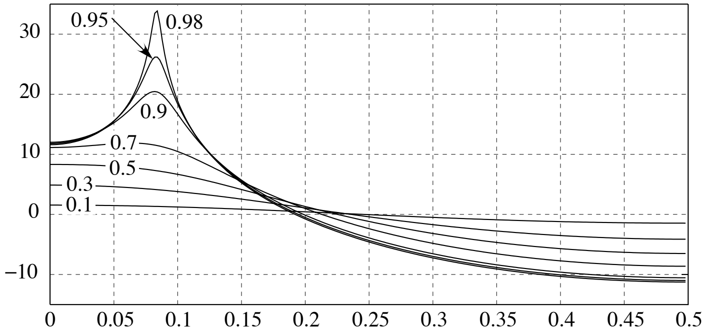


Figure H4.4 – *Gains of a second order filter as functions of the moduli of the poles*

3. The Jury test leads to: (i) $D(1) > 0 \Rightarrow 1 + a_1 + a_2 > 0$, (ii) n even $\Rightarrow D(-1) > 0 \Rightarrow 1 - a_1 + a_2 > 0$ and (iii) $|a_0| > |a_n| \Rightarrow 1 > |a_2|$. In the plane (a_1, a_2), these conditions delimit a triangle called the *stability triangle*.

H4.4 (Suppressing a sinusoidal component) (see page 125)

1. Type:

```
%===== CSUP50HZ1.M
%===== Frequency response of the rejection filter
nfft=256; freq=[0:nfft-1] / nfft;
phi=pi/4; ro=.9;
num=[ 1 -2*cos(phi) 1]; den=[ 1 -2*ro*cos(phi) ro*ro];
k=sum(den)/sum(num); num=k*num; % Normalization
snum=fft(num,nfft); sden=fft(den,nfft);
spec=snum ./ sden; plot(freq,abs(spec)); grid
```

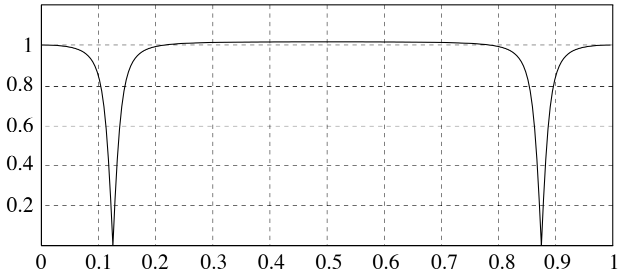


Figure H4.5 – *Gain of the rejection filter*

2. When the frequency f is very different from $\phi/2\pi$, the modulus is roughly equal to 1 and the phase is roughly equal to 0.

   In a neighborhood of ϕ (Figure H4.6), we have:

$$\begin{aligned} |H(f)| &\simeq \frac{MZ}{MP} \simeq \left|\frac{e^{2j\pi f}-e^{j\phi}}{e^{2j\pi f}-\rho e^{j\phi}}\right| = \left|\frac{2(1-\cos(2\pi f-\phi))}{1+\rho^2-2\rho\cos(2\pi f-\phi)}\right|^{1/2} \\ &\simeq \left|\frac{2\pi f-2\pi f_0}{1-\rho}\right| \end{aligned}$$

   And therefore:

$$|H(2j\pi f)| < \frac{1}{\sqrt{2}} \Rightarrow \Delta f \simeq \frac{1}{2\pi\sqrt{2}}(1-\rho)F_s$$

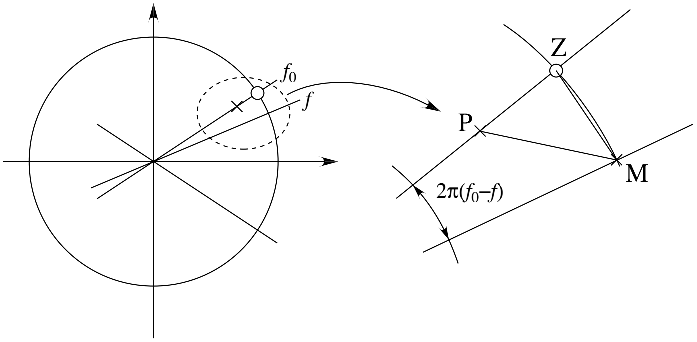


Figure H4.6 – *Construction of the rejection filter's frequency response*

3. Type:

```
%===== REJEC500HZ.M
[x,Fs]=wavread('phrase.wav');
N=length(x); x=x(:)/max(abs(x)); mtime=(0:N-1)';
Fb=500; % Hz
mnoise=sin(2*pi*Fb*mtime/Fs);
xnoisy=x+mnoise;
%====== rho is very close to 1. The transient part
% of the output is very long
rho=0.999; theta=2*pi*Fb/Fs; cost=cos(theta);
cosphi=cost*(1+rho*rho)/rho/2;
num1=0.5*[1+rho*rho -4*rho*cosphi 1+rho*rho];
den1=[1 -2*rho*cosphi rho*rho];
%=====
xdenoised=filter(num1,den1,xnoisy);
```

```
soundsc(xnoisy,Fs)
disp('Press a key'); pause
soundsc(xdenoised,Fs)
subplot(211); plot(xnoisy); subplot(212); plot(xdenoised)
```

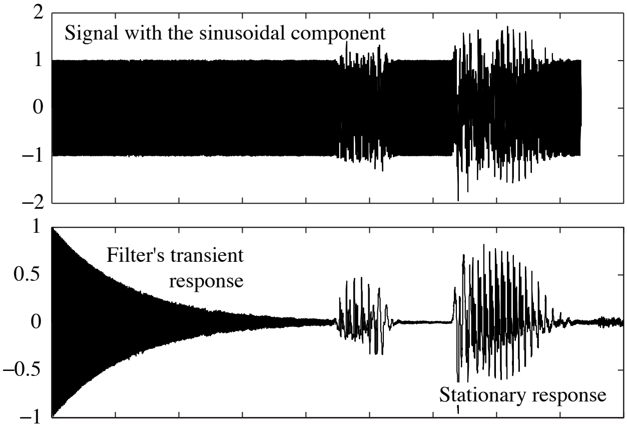


Figure H4.7 – *Effect of a value $\rho \approx 1$ on the filter's response*

4. Expression 4.30 is verified by writing that the numerator of 4.29 has the same zeros as 4.28, that is:

$$\frac{4\rho\cos\varphi}{1+\rho^2} = 2\cos\theta$$

The program **rejection.m** draws the complex gains and the poles and zeros of both transfer functions:

```
%===== REJECTION.M
nfft=1024; freq=[0:nfft-1]/nfft;
rho=[.8:.02:.99]; rho2=rho.*rho;
Lrho=length(rho);
theta=pi/4; cost=cos(theta);
figure(1); plot(fft([0 1],128)); grid;
set(gca,'AspectRatio',[2 1],'xlim',[-1 1],'ylim',[0 1])
hold on
num0=[1 -2*cost 1];
plot(roots(num0)+j*eps,'o');
for k=1:Lrho
    den1=[1 -2*rho(k)*cos(theta) rho2(k)];
    plot(roots(den1)+j*eps,'xr');
    num1=num0*sum(den1)/sum(num0);
    num2=(1+rho2(k))*[1 -2*cost 1]/2;
    den2=[1 -cost*(1+rho2(k)) rho2(k)];
    plot(roots(den2)+j*eps,'x');
    num1s=fft(num1,nfft); num2s=fft(num2,nfft);
    den1s=fft(den1,nfft); den2s=fft(den2,nfft);
    figure(2)
    plot(freq,abs(num1s./den1s)); grid on; hold on
    figure(3)
    plot(freq,abs(num2s./den2s)); grid on; hold on
    figure(1)
end
hold off
```

H4.5 (All-pass filter, properties of the maximum) (see page 130) Let $b_k = \rho e^{j\varphi}$. The transformation defined by:

$$P_k(z) = \frac{1 - b_k^* z}{z - b_k} = \frac{1-\rho^2}{z - b_k} - b_k^*$$

transforms the unit circle into itself.

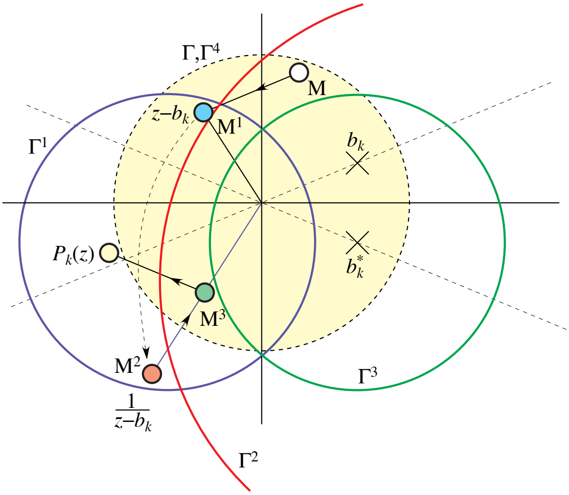


Figure H4.8 – *Construction of $P_k(z)$*

Let M (z) be a point inside the unit circle Γ. $z - b_k$ translates Γ to Γ^1. M is transformed into M^1 . The inversion $1/(z - b_k)$ takes M^2, the transformation of M^1, outside the circle Γ^2, transformation of Γ^1, because the inversion center remains inside Γ^1 ($|b_k| < 1$). The homothety and the translation that follows maintain the successive transformations of M outside of the transformed circles. In the end, the point $P_k(z)$ is therefore such that $|P_k(z)| > 1$.

Conversely, if M (z) is outside Γ, $P_k(z)$ will be inside Γ, hence $|P_k(z)| < 1$.

H4.6 (All-pass filter) (see page 130)

1. If we use the Parseval formula 2.26, then the filtering formula $Y(f) = H(f)X(f)$, we have:

$$\sum_{n=-\infty}^{+\infty} |y(n)|^2 = \int_{-1/2}^{1/2} |Y(f)|^2 df = \int_{-1/2}^{1/2} |H(f)|^2 |X(f)|^2 df$$

$$\Rightarrow \sum_{n=-\infty}^{+\infty} |y(n)|^2 = \sum_{n=-\infty}^{+\infty} |x(n)|^2$$

where we used the fact that $|H(f)| = 1$.

2. Let:

$$x_N(n) = \begin{cases} x(n) \text{ if } n \le N \\ 0 \text{ otherwise} \end{cases}$$

and let $y_N(n)$ be the filter's output signal. According to the result from the previous question:

$$\sum_{n=-\infty}^{N} |x(n)|^2 = \sum_{n=-\infty}^{+\infty} |x_N(n)|^2 = \sum_{n=-\infty}^{+\infty} |y_N(n)|^2$$

Because the filter is causal, $y_N(n)$ only depends on the values of $x_N(k)$ for $k \le n$ and therefore $y_N(n) = y(n)$ for $n \le N$. This means that:

$$\sum_{n=-\infty}^{+\infty} |y_N(n)|^2 = \sum_{n=-\infty}^{N} |y(n)|^2 + \underbrace{\sum_{n=N+1}^{+\infty} |y_N(n)|^2}_{\ge 0} \ge \sum_{n=-\infty}^{N} |y(n)|^2$$

Hence, for any N, $\sum_{n=-\infty}^{N} |x(n)|^2 \ge \sum_{n=-\infty}^{N} |y(n)|^2$.

H4.7 (Minimum phase filter) (see page 131)

1. Because the signal $x(n)$ is causal and because the filters $G_z(z)$ and $G_{zm}(z)$ are causal, the signals $y(n)$ and $y_m(n)$ are causal. Hence formula 14.12 can be applied, and we have:

$$\begin{cases} y(0) = \lim_{|z|\to+\infty} Y_z(z) = \lim_{|z|\to+\infty} G_z(z)X_z(z) \\ y_m(0) = \lim_{|z|\to+\infty} Y_{zm}(z) = \lim_{|z|\to+\infty} G_{zm}(z)X_z(z) \end{cases}$$

But the relation between the transfer function $G_{zm}(z)$ of the minimum phase filter and the transfer function of one of the filters with the same gain is of the type:

$$G_z(z) = G_{zm}(z)(1 - a^* z)/(z - a)$$

where $|a| < 1$. So if we impose $|z| \to +\infty$, we get:

$$|y(0)| = |y_m(0)||a| \leq |y_m(0)|$$

This result, shown for only one of the zeros, can of course be generalized to all of the zeros.

As a conclusion, the impulse response of the minimum phase filter is, from the very first value, more "intense" than any other filter with the same gain. Simply put, the minimum phase filter has a "quicker response".

2. Any filter $G_z(z)$ can be seen as the series cascade of the minimum phase filter $G_{zm}(z)$ and of an all-pass filter of the type:

$$H_z(z) = \prod_i (z^{-1} - a_i)/(1 - a_i z^{-1}) = \prod_i (1 - a_i^* z)/(z - a_i)$$

where $|a_i| < 1$. Because all the poles of $H_z(z)$ are inside the unit circle, the stable solution of $H_z(z)$ is *causal.* If we apply the result from question 2 of exercise 4.6, we can prove the expected result.

Therefore, among all the systems that have a frequency response with the same modulus, the minimum phase system is the one that transmits the most energy over the shortest period of time.

H4.8 (Window method: low-pass filter) (see page 146)

1. For N odd, we have:

$$h(n) = \int_{-f_0}^{f_0} e^{2j\pi n f} df = \frac{\sin(2\pi n f_0)}{\pi n}$$

and for N even:

$$h(n) = \int_{-f_0}^{f_0} e^{j\pi f} e^{2j\pi n f} df = \frac{\sin(2\pi(n+1/2)f_0)}{\pi(n+1/2)}$$

2. Type:

```
function h=rif(N,f0)
%%========================================================%
%% FIR synthesis using the window method (Hamming window) %
%% SYNOPSIS: h=RIF(N,f0)                                  %
%%    h  = Length N impulse response                      %
%%    N  = Filter order                                   %
%%    f0 = Normalized cut-off frequency                   %
%%========================================================%
P=fix(N/2); ham=0.54-0.46*cos(2*pi*(0:P-1)/(N-1));
if (rem(N,2)==0)              % N even
    d=((-P:-1)+.5)*pi; h=sin(2*d*f0) ./ d;
    h=h .* ham ; h=[h h(P:-1:1)];
else                          % N odd
    d=(-P:-1)*pi; h=sin(2*d*f0) ./ d;
    h=h .* ham ; h=[h 2*f0 h(P:-1:1)];
end
return
```

H4.9 (Spectrum reversal encryption) (see page 146)

1. The spectrum $Y(F)$ can be written:

   $$Y(F) = X(F + F_m) + X(F - F_m)$$

   If $X(F) = 0$ for $|F| > B$, a low-pass filtering $(-F_m, +F_m)$ leads to the expected spectrum. The decryption operation is the same as the encryption operation, that is a multiplication by $2 \times \cos(2\pi F_m t)$ followed by a low-pass filtering of the $(-B, +B)$ band. Because of the shape of the encrypted signal, this is sometimes called “spectrum reversal”.

2. Sample a crypted sound (such as the sound on certain television channels). Let us assume that $B = 10$ kHz and $F_m = 12{,}8$ kHz. Type:

```
%===== DECOCPLUS.M
Fs=48000; Fm=12800; B=10000;
load soncrypt.dat; N=length(soncrypt);
fm=Fm/Fs; b=B/Fs;
y=soncrypt .* cos(2*pi*fm*(1:N));
hh=rif(31,b); z=filter(hh,1,y); soundsc(z,Fs)
```

H4.10 (Window method: band-pass filter) (see page 147)

1. The DTFT of the sequence $2h(n)\cos(2\pi n f_0) = h(n)e^{2j\pi n f_0} + h(n)e^{-2j\pi n f_0}$ can be written as $H(f - f_0) + H(f + f_0)$. Hence, if $H(f)$ is the complex gain of a low-pass filter, the result is a filter the complex gain of which is centered around $\pm f_0$, therefore a band-pass filter.

2. Type:

```
%===== CFENPBANDE.M
%===== Band-pass filter
Lfft=1024; fq=(0:Lfft-1)/Lfft; f0=0.2; fb=0.1;
N=input('length: ');
P=fix(N/2); R=rem(N,2); h=rif(N,fb/2);
if (R==0), D=(-P:P-1)+1/2; else, D=(-P:P); end
g=2*h .* cos(2*pi*f0*D); gf=fft(g,Lfft);
agf=abs(gf); phigf=angle(gf);
figure(1); plot(fq(1:Lfft/2),agf(1:Lfft/2)); grid
figure(2); plot(fq(1:Lfft/2),phigf(1:Lfft/2)); grid
```

Notice that if we want to maintain a linear phase, we have to multiply by $2\cos(2\pi n f_0)$ if N is odd and by $2\cos(2\pi(n+1/2)f_0)$ if N is even.

Figure H4.9 shows the gabarit obtained for $N = 80$.

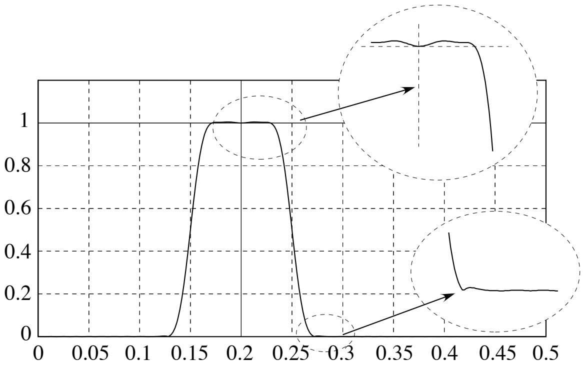


Figure H4.9 – *Spectrum of the band-pass filter*

H4.11 (Window method: derivative filter) (see page 147)

1. The Fourier transform of $dx(t)/dt$ is $2j\pi F X(F)$ (page 728). This expression corresponds to the filtering of $x(t)$ by a filter with the complex gain $H_a(F) = 2j\pi F$. This expression calls for a comment: in the case of $(-F_s/2, +F_s/2)$ band-limited real signals, the method given on page 133 provides, for the digital filter, a complex gain equal to $H(f) = 2j\pi F_s f$ in the band $(-1/2, 1/2)$.

2. The previous results lead us to the Fourier series expansion coefficients of $H(f)$, which have the expression:

$$h(n) = F_s \int_{-1/2}^{1/2} 2j\pi f e^{2j\pi f} df$$

In theory, a derivative filter is such that the output dimension is the same as the input dimension divided by seconds, or in other words multiplied by Hertz. This explains how the term F_s appears in the digital filter's impulse response. When all the calculations are done, we get:

$$h(n) = \begin{cases} 0 & \text{for} \quad n = 0 \\ F_s \dfrac{\cos(\pi n)}{n} & \text{for} \quad n \neq 0 \end{cases} \tag{13.3}$$

The sequence is truncated between $-N$ and $+N$, and the result is multiplied by the weighting function.

3. For $N = 12$, the filter's output must have the expression:

$$u(n) = h(-12)x(n+12) + \cdots + h(0)x(n) + \cdots + h(12)x(n-12)$$

This is not a causal solution: it requires the 12 future input values to be known in order to calculate the output $u(n)$. A causal realization consists of taking:

$$y(n) = h(-12)x(n) + \cdots + h(0)x(n-12) + \cdots + h(12)x(n-24)$$

hence $y(n) = u(n-12)$. This solution causes a 12 sample delay.

4. The derivative function is:

```
function [y,hder]=deriv(N,x,Fs)
%%=========================================%
%% Digital derivation                      %
%% SYNOPSIS: [y,hder]=DERIV(N,x,Fs)        %
%%    N    = (2N+1) coefficient FIR filter  %
%%    x    = Signal                        %
%%    Fs   = Sampling frequency (default: 1) %
%%    y    = result                        %
%%    hder = Impulse response of the filter  %
%%=========================================%
if nargin<3, Fs=1; end;
if nargin<2, error('Parameters are missing.'); return; end
hder=cos(pi*(1:N)) ./ (1:N); hder=[-fliplr(hder) 0 hder];
%===== Hamming window
hder=Fs * hder .* (0.54+0.46*cos((-N:N)*pi/N));
y=filter(hder,1,x);
return
```

The following program tests the derivative filter on the function $x_a(t) = \sin(2\pi F_0 t)$. Notice the shift due to the causal design, as well as the transient state due to the choice of the initial conditions. Type:

```
%===== CDERSIN.M
N=12; Fs=4000; nfft=512; freq=(0:nfft-1)/nfft*Fs;
F0=300; T=100; t=(0:T-1)/Fs;
%===== Original
x=sin(2*pi*F0*t);
subplot(321); plot(t,x); grid
axis([0 (T-1)/Fs -1.2 1.2]); title('x(t)')
%===== Theoretical derivative
xp=2*pi*F0*cos(2*pi*F0*t);    % result
ordm=1.2*2*pi*F0;
subplot(322); plot(t,xp); grid
axis([0 (T-1)/Fs -ordm ordm]); title('x''(t)')
%===== Digital derivative
[y hder]=deriv(N,x); y=Fs*y;
subplot(323); plot(t,y); grid
axis([0 (T-1)/Fs -ordm ordm]); title('y(t)')
%===== Delay due to the filter
subplot(324); plot(t,xp,'b',t-N/Fs,y); grid
axis([0 (T-1)/Fs -ordm ordm]); title('y(t-N/Fs)')
%===== Gain of the derivative filter
hders=fft(Fs*hder,nfft);
subplot(313)
plot(freq,abs(hders),[0 Fs/2],[0 Fs*pi]);
axtemp=axis; axis([0 Fs/2 axtemp(3:4)]); grid
```

This program also provides the gain of the derivative filter and of the obtained filter. Try both the rectangular window and the Hamming window. Check that the amplitude ratio of $x(t)$ to $y(t)$ is equal to the value of the gain in $F_0 = 300$ Hz.

The following program tests the derivative filter on a *periodic square* signal (Figure H4.10):

```
%===== CDERHOR.M
clear; N=30; Fs=100;
xT=[ones(1,50) zeros(1,50)]; x=[xT xT xT xT];
T=length(x); t=(0:T-1)/Fs;
[y hder]=deriv(N,x); y=y*Fs;
%===== Original signal
subplot(311); plot(t,x);
axis([0 T/Fs -1.5 1.5]); grid
%===== Impulse response of the filter
subplot(312);
plot([0:2*N],hder,'-',[0:2*N],hder,'o'); grid
%===== Result
subplot(313); plot(t-N/Fs,y);
axis([0 T/Fs -Fs Fs]); grid
```

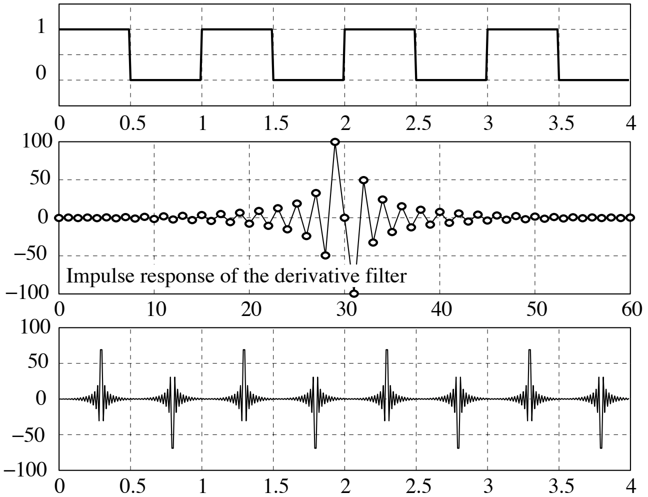


Figure H4.10 – *Differentiation of a periodic square signal*

H4.12 (Butterworth filter) (see page 149)

1. Type:

```
%===== CBUTTER1.M
n=input('Butterworth filter order: ');
nr=rem(n,2); den=[1 nr zeros(1,n-1)];
nb=(n - nr)/2;
for k=1:nb
    alp=pi*(2*k-1+nr)/n/2; den2=[1 2*cos(alp) 1];
    den=filter(den2,1,den);
end
den       % Displaying the result
```

2. Type:

```
%===== CBUTTER2.M
Nordre=6; mmax=40; omeg=[1:mmax] * 0.1;
y=j * omeg; g=zeros(mmax,Nordre-1);
for iord=2:Nordre
   nr=rem(iord,2); den=[1 nr zeros(1,iord-1)];
   nb=(iord - nr)/2;
%===== Generating the denominator
   for k=1:nb
      alp=pi * (2*k-1+nr)/iord/2;
      den2=[1 2*cos(alp) 1]; den=filter(den2,[1],den);
   end
%===== mmax points
```

```
   for k=1:mmax
      mm=triu(ones(iord+1),1) * y(k) + tril(ones(iord+1),0);
      g(k,iord-1)=1/(prod(mm)*den');
   end
end
loglog(omeg,abs(g)); grid;
```

3. Bilinear transformation program:

   (a) Horner representation of the polynomial $g(x)$:

   $$\begin{cases} g_0 = a_n \\ g_k(x) = a_{N-k} + g_{k-1}(x)x \quad \text{for} \quad k = 1 : N \end{cases}$$

   (b) Rational function variable change: $x = B(z)/A(z)$. If we define $g_k = N_k/D_k$, we get for $k = 1, \ldots, N$:

   $$N_0(z) = a_n \text{ and } D_0(z) = 1$$

   $$k = 1, \ldots N : \begin{cases} D_k(z) = D_{k-1}(z)A(z) \\ N_k(z) = a_{N-k}D_k(z) + N_{k-1}(z)B(z) \end{cases}$$

   (c) For the bilinear transform (in our case choose $T = 1$):

   $$x = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}$$

4. Type:

```
function [B,A]=nbilin(pol,Ts)
%%==================================================%
%% Bilinear transform of a polynomial               %
%% SYNOPSIS: [B,A]=NBILIN(pol,Ts)                   %
%%    pol = Polynomial (decreasing powers of s)     %
%%        = a0 s^n+a1 s^(n-1)+ ... +aN              %
%%    Ts  = Sampling period                         %
%%    B,A = Numerator and denominator of the result %
%%==================================================%
if nargin<2, Ts=1; end
NX=[1 -1]*2/Ts; DX=[1 1];
nP=length(pol); PP=zeros(nP,1); PP(:)=pol;
B=pol(1); A=[1];
for k=2:nP
    A=conv(A,DX); B=conv(B,NX) + pol(k)*A;
end
return
```

H4.13 (Temporal aliasing and the use of the DFT) (see page 150)
Let us calculate the original $\tilde{h}(n)$ of $\{H(k/N)\}$:

$$\begin{aligned}\tilde{h}(n) &= \frac{1}{N}\sum_{k=0}^{N-1} H(k/N)e^{2\pi j\frac{nk}{N}} = \frac{1}{N}\sum_{k=0}^{N-1}\left(\sum_{m=-\infty}^{\infty} h(m)e^{-2\pi j\frac{mk}{N}}\right)e^{2\pi j\frac{nk}{N}} \\ &= \frac{1}{N}\sum_{m=-\infty}^{\infty} h(m)\sum_{k=0}^{N-1} e^{-2\pi j\frac{(m-n)k}{N}}\end{aligned}$$

The last sum is different from zero, and equal to N when $m = n[N]$. We get:

$$\tilde{h}(n) = \sum_{r=-\infty}^{\infty} h(n+rN), \text{ with } n \in [0, N-1]$$

This sum expresses a *temporal aliasing* phenomenon, which is negligible if the number of points chosen for "sampling" $H(f)$ is large. For example, in the case of a low-pass filter, the coefficients behave like $1/n$ and the temporal aliasing grows fainter as N increases:

```
%===== REPLIEMTEMP.M
%===== Window method
nfft=1024; freq=[0:nfft-1]/nfft;
fc=.1;
Ng=128; Npts=Ng/2; n=[-Npts:Npts-1]; tps=n+Npts;
pit=((-Npts:Npts-1)+.5)*pi;
hn=sin(2*pit*fc) ./ pit; hns=fft(hn,nfft);
hnf=hn .* (.54-.46*cos(pi*tps/Npts)); hnfs=fft(hnf,nfft);
subplot(221); stem(n,hnf); grid
Mx=max(hnf)*1.5; Mn=min(hnf)*1.2;
set(gca,'ylim',[Mn Mx])
subplot(212); plot(freq,abs(hnfs),'-',freq,abs(hns),'-g'); grid
%===== Direct method
Ngt=128; frt=[0:Ngt-1]/Ngt;
fcd=fix(fc*Ngt)+1; Nz=Ngt-2*fcd;
Gfk=[ones(1,fcd) zeros(1,Nz+1) ones(1,fcd-1)];
subplot(212); hold on; plot(frt,Gfk, 'or'); hold off;
set(gca,'xlim',[0 .5])
%set(gca,'xlim',[.5 1.5]*fc,'ylim',[-0.05 1.3])
%===== Calculating hnt
hnt=real(ifft(Gfk)); hnt=[hnt(Ngt/2+1:Ngt) hnt(1:Ngt/2)];
subplot(222); stem([-Ngt/2:Ngt/2-1], hnt); grid
set(gca,'ylim',[Mn Mx])
%===== Windowing the hnt
hntf=hnt .* (.54-.46*cos(2*pi*[0:Ngt-1]/(Ngt-1)));
%===== Verification with the spectrum
```

```
nfft=8192; freq=[0:nfft-1]/nfft;
hnts=fft(hnt,nfft); hntfs=fft(hntf,nfft);
subplot(212); hold on;
plot(freq,abs(hnts),'b',freq,abs(hntfs),'r'); hold off;
set(gca,'xlim',[fc*.8 fc*1.2])
```

H4.14 (Interpolation) (see page 152)

1. The `interM.m` function interpolates by a factor M. To approximate the ideal low-pass filter with the band $(-1/(2M), +1/(2M))$ and the gain 1, we used the window method for the computation of the lenght 81 FIR filter. The window used is a Hamming window:

```
function y=interM(x,M,Nf)
%%===============================%
%% Interpolation function        %
%% SYNOPSIS: y=INTERM(x,M,Nf)    %
%%       x  = Input sequence     %
%%       M  = Interpolation ratio  %
%%       Nf = 2Nf+1 coeffts filter %
%%       y  = Output sequence    %
%%===============================%
if nargin<3, Nf=40; end
%===== Low-pass filter
theta=pi*[1:Nf]; h=sin(theta/M) ./ (theta);
%===== Hamming window
h=h .*  (.54 + .46*cos(theta/Nf));
h=[fliplr(h) 1/M h]; h=h/sum(h)*M;
%===== Insertion of zeros
x0=zeros(length(x)*M+Nf,1); x0(1:M:length(x)*M)=x;
y =filter(h,1,x0); y=y(Nf+1:length(y));
return
```

2. An application example:

```
%===== INTERMEX.M
x=rand(1,40);
M=4; y=interM(x,M);
plot(y); hold on; plot(y,'xr');
plot([1:M:length(y)],x,'o'); hold off
grid
```

H4.15 (Undersampling) (see page 156)

1. The `decM.m` function undersamples by a factor M. To approximate the ideal low-pass filter with the band $(-1/(2M), +1/(2M))$ and the gain 1, we used the window method for the computation of a FIR filter:

```
function y=decM(x,M,Nf)
%%===============================%
%% Decimation function           %
%% SYNOPSIS: y=DECM(x,M,Nf)      %
%%      x  = Input sequence      %
%%      M  = Decimation ratio    %
%%      Nf = 2Nf+1 coeffts filter %
%%      y  = Output sequence     %
%%===============================%
if nargin<3, Nf=20; end
theta=(1:Nf)*pi; h=sin(theta/M) ./ (theta/M);
h=h .* (0.54 + 0.46 * cos(theta/Nf)); % Hamming window
h=[fliplr(h) 1 h]/M;
x0=zeros(length(x)+Nf,1); x0(1:length(x))=x;
y= filter(h, 1, x0);
y=y(Nf+1:M:length(y));  % Decimation
return
```

2. Undersampling a speech signal:

```
%===== DECMPAROLE.M
load phrase
Lsn=length(sn);
soundsc(sn,8000)                      % Original signal
%===== One out of every 2 samples
sn2=sn(1:2:Lsn); soundsc(sn2,4000)
sn2se=decM(sn,2); soundsc(sn2se,4000) % Undersampling with M=2
```

H4.16 (Paralleled undersampling and oversampling) (see page 156)

1. Figure H4.11 shows the design structure of the factor M oversampler: the signal we wish to undersample is "broken up" into M delayed and undersampled signals that are filtered in parallel by M filters. If the ideal low-pass filter is approximated by a length $L = \ell M$ filter, each filter in the diagram H4.11 has a length ℓ.

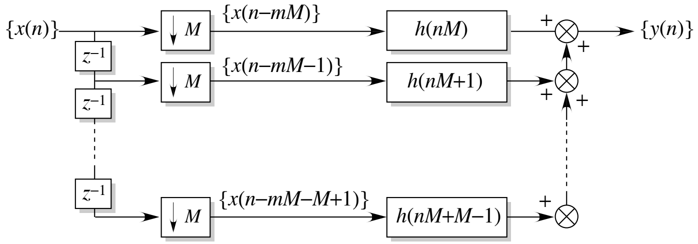


Figure H4.11 – *Polyphase architecture of undersampling*

2. For $M = 4$ and for a length 8 filter, let us write the output signal $y(n)$. In $n = 0$, we have $y(0) = h(0)x(4n)$. For $n = 1$, $y(4) = (h(0)x(4) + h(4)x(0)) + h(1)x(3) + h(2)x(2) + h(3)x(1)$. And for $n > 1$, we get:

$$\begin{aligned} y(4n) &= (h(0)x(4n) + h(4)x(4n-4)) \\ &+ (h(1)x(4n-1) + h(5)x(4n-5)) \\ &+ (h(2)x(4n-2) + h(6)x(4n-6)) \\ &+ (h(3)x(4n-3) + h(7)x(4n-7)) \end{aligned}$$

Therefore, $y(4n)$ is the sum of 4 filterings involving the sequences $\{x(0), x(4)\dots\}$, $\{0, x(3), x(7)\dots\}$, $\{0, x(2), x(6)\dots\}$ and $\{0, x(1), x(5)\dots\}$. This computation is performed in the following program:

```
%===== DECPARA.M
clear; M=4;
N=1500; L=16; % N and M must be multiple of M
x=randn(N,1); h=(1:L);
%===== Direct undersampling -> yu
y=filter(h,1,x); yu=y(1:M:N);
%===== Parallelized undersampling -> yp
yp=zeros(N/M,1);
for k=1:M
    auxx=x(k+1:M:N); lx=length(auxx); auxh=h(M-k+1:M:end);
    yp(1:lx)=yp(1:lx)+filter(auxh,1,auxx);
end
max(abs(yu(M+1:lx)-yp(M:lx-1)))
```

3. Type:

```
%===== OVERPARA.M
clear all
M=4; N=150; L=16;
x=randn(N,1); h=(1:L);
xo=zeros(N*M,1); xo(1:M:end)=x;
%===== Direct oversampling -> yo
yo=filter(h,1,xo);
yp=zeros(N*M,1);
%===== Parallelized oversampling -> yp
for k=1:M
    auxh=h(k:M:end);
    yp(k:M:N*M)=yp(k:M:N*M)+filter(auxh,1,x);
end
max(abs(yo(M:end)-yp(M:end)))
```

H5 Filter implementation

H5.1 (Filter architecture) (see page 163)

1. We can write:

$$\left\{\begin{array}{lcl} x_p(n) & = & b_p i(n) - a_p o(n) \\ x_{p-1}(n) & = & b_{p-1} i(n) - a_{p-1} o(n) + x_p(n-1) \\ \vdots & & \vdots \\ x_1(n) & = & b_1 i(n) - a_1 o(n) + x_2(n-1) \\ 0 & = & b_0 i(n) - o(n) + x_1(n-1) \end{array}\right.$$

$$\Rightarrow \left\{\begin{array}{l} x_1(n) = b_1 i(n) - a_1 o(n) + b_2 i(n-1) - \cdots \\ \qquad\qquad \cdots + b_p i(n-p+1) - a_p o(n-p+1) \\ x_1(n-1) + b_0 i(n) = o(n) \end{array}\right.$$

In terms of the z-transform, we get, as expected, the transfer function:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_p z^{-p}}{1 + a_1 z^{-1} + \cdots + a_p z^{-p}}$$

2. The state representation associated with this architecture is:

$$\left\{\begin{array}{l} \begin{bmatrix} x_1(n) \\ \vdots \\ x_p(n) \end{bmatrix} = \begin{bmatrix} -a_1 & 1 & 0 & \cdots & 0 \\ -a_2 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \vdots & \vdots & & \ddots & 1 \\ -a_p & 0 & \cdots & \cdots & 0 \end{bmatrix} \begin{bmatrix} x_1(n-1) \\ \vdots \\ x_p(n-1) \end{bmatrix} + \begin{bmatrix} b_1 - b_0 a_1 \\ \vdots \\ b_p - b_0 a_p \end{bmatrix} i(n) \\ o(n) = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix} \mathbf{x}(n-1) + b_0 i(n) \end{array}\right.$$

The corresponding filtering program is given below. Of course, this design is far from being the optimal one in terms of execution. It would be preferable to have a "**mex**" function. You can check that it leads to the same output sequence as the one obtained with the filtering function described in the text:

```
function [xout,zs]=filtrerII(num,den,xinp,zi)
%%===============================================%
%% Filtering (Transpose-form IIR filter structure) %
%% SYNOPSIS: [xout,zs]=FILTRERII(num,den,xinp,zi)  %
%%        num  = [b0 b1 ... bP]                     %
%%        den  = [1 a1 a2 ... aP]                   %
%%        xinp = Input sequence                     %
%%        zi   = Initial state                      %
%%        xout = Output sequence                    %
%%        zs   = Final state                        %
%%==================================================%
lden=length(den); lnum=length(num);
if lden < lnum, den(lnum)=0; lden=lnum; end
if lnum < lden, num(lden)=0; end
ld=lden-1; N=length(xinp); av=zeros(ld,1); bv=av;
av(:)=den(2:lden); bv(:)=num(2:lden);
if nargin==3, zi=zeros(ld,1); end;
if length(zi)<ld, zi(ld)=0; end
zzi=zeros(ld,1); zzi(:)=zi; zs=zzi;
%===== State representation
b0=num(1); ma=compan([1;av])';
vb=bv - b0 * av; vc=[1 zeros(1,ld-1)]; cd=b0;
%===== Filtering
for ii=1:N,
    zsn =ma * zs + vb * xinp(ii);
    xout(ii)=vc * zs + cd * xinp(ii); zs=zsn;
end
return
```

3. We can express the intial state reconstruction by:

$$x_k(0) = -\sum_{\alpha=0}^{k-1} b_\alpha i(k-\alpha) + \sum_{\alpha=0}^{k-1} a_\alpha o(k-\alpha)$$

This leads us to the state reconstruction program:

```
function zi=filtricII(num,den,xinp,xout)
%%===========================================%
%% Reconstruction of the initial state for a %
%% Transpose-Form IIR structure              %
%% SYNOPSIS: zi=FILTRICII(num,den,xinp,xout) %
%%        num  = [b0 b1 ... bP]              %
%%        den  = [1 a1 a2 ... aP]            %
%%        xinp = Input sequence              %
%%        xout = Output sequence             %
%%        zi   = Reconstructed initial state %
%%===========================================%
lden = length(den); lnum = length(num);
if lden<lnum, den(lnum)=0; lden=lnum; end
if lnum<lden, num(lden)=0; end
ld=lden-1; numv=zeros(lden,1); denv=numv;
```

```
numv(:)=num; denv(:)=den;
%=====
lx = length(xinp); ly = length(xout);
if lx<ld, xinp(ld)=0; end
if ly<ld, xout(ld)=0; end
ysv=zeros(1,ld); xev=ysv; ysv(:)=xout(1:ld);
xev(:)=xinp(1:ld);
zi=filtrerII(denv,1,ysv)+filtrerII(-numv,1,xev);
return
```

H5.2 (Parallel implementation of the FIR filtering) (see page 164)
Type:

```
%===== POLYPHASE.M
x0=[1:103]; lx0=length(x0); M=4;
b=0.3; N=25; h=rif(N,b);
%===== M-polyphase filters (with insertion of zeros
%     for the processing)
hp=zeros(M,N);
for k=1:M, hp(k,1:M:N-k+1)=h(k:M:N); end
%===== Result of the filtering without polyphase
z1=filter(h,1,x0);
%===== Polyphase processing
z2=zeros(M,lx0);
for k=1:M,
    xx = [zeros(1,k-1) x0(1:lx0-k+1)];
    z2(k,:)=filter(hp(k,:),1,xx);
end
xx = sum(z2); [xx(1:lx0)' z1(1:lx0)']
```

H5.3 (FFT filtering) (see page 172)

1. The gain at the frequency 0 is equal to the sum of the impulse response coefficients.

2. Type:

```
%===== FILTRAGEFFT1.M
nfft=256; freq=[0:nfft-1]/nfft;
hn=[0.0002 0.0134 0.0689 0.1676 0.2498 ...
    0.2498 0.1676 0.0689 0.0134 0.0002];
nh = length(hn);
N=128-nh; temps=[0:N-1]; f0=.15; f1=.3;
x=sin(2*pi*f0*temps) + sin(2*pi*f1*temps);
%===== Processing using the convolution
hn = hn / sum(hn); y=filter(hn,1,x);
subplot(311); plot(temps,[x' y'])
subplot(312); plot(freq, abs(fft(hn,nfft)));
subplot(313); plot(freq, abs(fft([x' y'],nfft)));
```

3. Then type:

```
%===== FILTRAGEFFT2.M
% ,---------,--------------------------------------,
% ! nh*"0"  ! x(0) x(1)            ... x(n)  x(N-1)!
% '---------'--------------------------------------'
% ,---------,--------------------------------------,
% !           ...     0 h(nh-1)    ... h(0) 0 ...  !
% '---------'--------------------------------------'
xcompl = [zeros(1,nh) x]; nxc = length(xcompl);
hns = fft(hn,nxc); xcs = fft(xcompl,nxc);
yns = xcs .* hns; yn = real(ifft(yns));
figure(3)
plot(y); hold; plot(yn(1+nh:nxc),'or'); hold;
```

4. Then type:

```
%===== FILTRAGEFFT3.M
% Processing with size P blocks
clear hns; clear yns; P=32; hns = fft(hn,P);
kp=floor(nxc / (P-nh));        % Number of blocks
y=[];
for k =0:kp-1,
    kdb=(P-nh)*k;
    xbloc=xcompl(kdb+1:kdb+P); % "overlap"
    xbs=fft(xbloc,P); yns=hns .* xbs;
    yn = real(ifft(yns)); y=[y yn(nh+1:P)];
end
hold; plot(y,'ob'); hold; grid
```

H5.4 (Band-pass filter based on a comb filter) (see page 175)
A pole was placed in the first cell of the low-pass filter from Figure 5.10 in order to cancel the zero placed at the frequency 0 in the second cell. Therefore, all we need to do to design a real pass-band filter around the frequency m/M is to precede the filter with another filter with the transfer function:

$$F_m(z) = \frac{1}{(1-w_m z^{-1})(1-w_m^* z^{-1})} = \frac{1}{1-2\cos(2\pi m/M)z^{-1}+z^{-2}}$$

This leads to $H_z(z) = F_m(z)(1-z^{-M})$ which is still an FIR filter. Figure H5.1 shows, for $M = 16$, the frequency response of the low-pass filter and of the band-pass filter for $m = 3$.

The following program plots the frequency responses of the two filters:

```
%===== PEIGNE.M
M=16; m=(0:M-1); Lfft=512; fq=(0:Lfft-1)/Lfft-1/2;
fq1=(Lfft/2+1:Lfft);fq2=(1:Lfft/2);
%===== Low-pass and band-pass comb filters
```

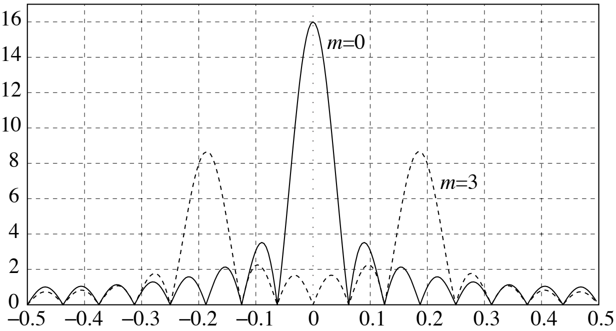


Figure H5.1 – *Frequency response: $m = 0$ (low-pass) and $m = 3$ (band-pass)*

```
ht=ones(1,M); k=4; gt=2 * ht .* cos(2*pi*k*m/M);
hf=abs(fft(ht,Lfft)); hf=10*log10(hf / max(abs(hf)));
gf=abs(fft(gt,Lfft)); gf=10*log10(gf / max(abs(gf)));
subplot(211); plot(fq,[hf(fq1) hf(fq2)]);
set(gca,'ylim',[-20 0]); grid
subplot(212); plot(fq,[gf(fq1) gf(fq2)]);
set(gca,'ylim',[-20 0]); grid
```

H6 An Introduction to image processing

H6.1 (Logical functions) (see page 195)

1. Logical functions:

```
function pixr=FoncLog(pix1,ffl,pix2)
%%===================================================%
%% Logical operation                                 %
%% SYNOPSIS: pixr=FONCLOG(pix1,ffl,pix2)             %
%%    pix1 = First image (gray levels)               %
%%    ffl  = Logical function '&', '|', 'xor', '~', %
%%           '==', '<', '>', '<=', '>=', '==', '~=' %
%%           elsewhere pixr=~(pix1)                  %
%%    pix2 = Second image (gray levels)              %
%%    pixr = Resulting image                         %
%%===================================================%
cm=colormap; L=log(size(cm,1))/log(2); N1=size(pix1);
if (nargin<3),
    ffl='~';
else
    if (N1 ~= size(pix2)),
        error('Matrix dimensions have to match!')
    end
```

```
end
pixr=zeros(N1);
%===== Analysis of the logical function
if findstr('~',ffl),
   chaine='~(rem(pix1,2))';
elseif findstr('&|<<=>>==~=',ffl),
   chaine=['rem(pix1,2) ' ffl ' rem(pix2,2)'];
elseif (findstr(ffl,'xor')),
   chaine='xor(rem(pix1,2),rem(pix2,2))';
else,
   chaine='~(rem(pix1,2))'; %
end
%===== Processing
for k=1:L
    pixr=pixr+ eval(chaine) * 2^(k-1); pix1=fix(pix1/2);
    if (nargin==3), pix2 = fix(pix2/2); end
end
return
```

2. Test program for the `FoncLog` function:

```
%===== TESTLOGIC2.M
load lena25;                  % Loading and displaying
subplot(221); imagesc(pixc);  % the first image
colormap(cmap); axis('image');
set(gca,'units','pixels')
title('Original')
%===== Loading and displaying the second image
load testlog1;
subplot(222); imagesc(pixtl); axis('image')
title('Mask')
%===== Logical operators >= and ~
pixr = FoncLog(pixc,'>=',pixtl);
subplot(223); imagesc(pixr); axis('image');
title('Result of the >=')
pixr = FoncLog(pixc);
subplot(224); imagesc(pixr); axis('image');
title('Result of the ~')
```

H6.2 (Plane transformation) (see page 198)

1. Linear transformation:

```
function pixcR=lintrimg(pixc,M)
%%==================================%
%% Linear transform of an image     %
%% SYNOPSIS: pixcR=LINTRIMG(pixc,M) %
%%   pixc  = Image (nl*nc) pixels   %
%%      M  = Transform matrix       %
%%   pixcR = Result                %
%%=================================%
Spix=size(pixc); Nl=Spix(1); Nc=Spix(2);
%=====. Pixel coordinates
tbx=ones(Nl,1)*[1:Nc]; tby=[1:Nl]'*ones(1,Nc);
nlg=reshape(tby,1,Nl*Nc); ncl=reshape(tbx,1,Nl*Nc);
tbidx=[nlg;ncl];
idtb=nlg+(ncl-1)*Nl; % Linear Indices
%===== Transformation
tbxy=[ncl;Nl-nlg];   % Coordinates
tbv=round(M*tbxy);
xmin=min(tbv(1,:)); xmax=max(tbv(1,:)); ncol=xmax-xmin+1;
ymin=min(tbv(2,:)); ymax=max(tbv(2,:)); nlig=ymax-ymin+1;
pixcR=zeros(nlig,ncol);
tbidxR=[nlig-(tbv(2,:)-ymin);tbv(1,:)-xmin+1];
idtbR=tbidxR(1,:)+(tbidxR(2,:)-1)*nlig; % Linear Indices
pixcR(idtbR)=pixc(idtb);
return
```

2. Defining the transformation (Figure H6.1), type:

```
%===== TRANSFTRI.M
% From one triangle to another using (x,y)-coordinates
h1=figure; set(h1,'color',[1 1 0])
set(gca,'xlim',[0 100],'ylim',[0 100],'NextPlot','add');
grid
trixy=[];
for k=1:3
    pt=ginput(1); trixy=[trixy;pt]; plot(pt(1),pt(2),'ob')
    text(pt(1)-1,pt(2)+3,int2str(k))
end
plot([trixy(:,1);trixy(1,1)],[trixy(:,2);trixy(1,2)])
triXY=[];
for k=1:3
    pt=ginput(1); triXY=[triXY;pt]; plot(pt(1),pt(2),'or')
    text(pt(1)-1,pt(2)+3,int2str(k))
end
plot([triXY(:,1);triXY(1,1)],[triXY(:,2);triXY(1,2)],'r')
%===== Transformation matrix
Mxy=[trixy [1;1;1]]'; MXY=[triXY [1;1;1]]';
M=MXY*inv(Mxy)
```

3. Applying the transformation (Figure H6.2) can lead to many “missing points” that have to be processed. In the example we chose, a simple median filtering is enough, but this is rarely the case:

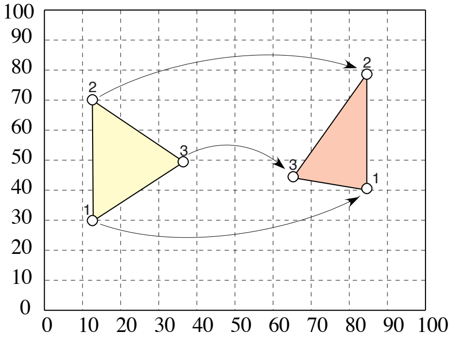


Figure H6.1 – *Defining the transformation*

```
%===== TRAITTRANSFTRI
load clah % Loading the image, the palette and
          % the transformation matrix
set(gcf,'color',[1 1 1]);
xor=40; yor=40; % Position in the window
subplot(121); imagesc(pixc0); colormap(cmap)
Spix=size(pixc0); Nl=Spix(1); Nc=Spix(2);
set(gca,'units','pixels','Position',[xor yor Nc Nl]);
%===== Linear transform
pixcR=lintrimg(pixc0,M(1:2,1:2));
%===== Display
imageS = median2D(pixcR,3,3);
Spix=size(pixcR); nlig=Spix(1); ncol=Spix(2);
subplot(122); imagesc(imageS); colormap(cmap)
set(gca,'units','pixels','Pos',[xor+xor+Nc yor ncol nlig]);
```

H6.3 (Transformation of a rectangular selection) (see page 198)
We can write:

$$\begin{bmatrix} X_1 \\ Y_1 \\ X_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1Y_1 & -y_1Y_1 \\ x_1 & y_1 & \cdots & & & & & \\ \vdots & \vdots & & & & & & \\ \vdots & \vdots & & & & & & \\ \vdots & \vdots & & & & & & \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4Y_4 & -y_4Y_4 \end{bmatrix} = \begin{bmatrix} a \\ b \\ t_x \\ c \\ d \\ t_y \\ e \\ f \end{bmatrix}$$

By solving this system, we can determine the transformation matrix:

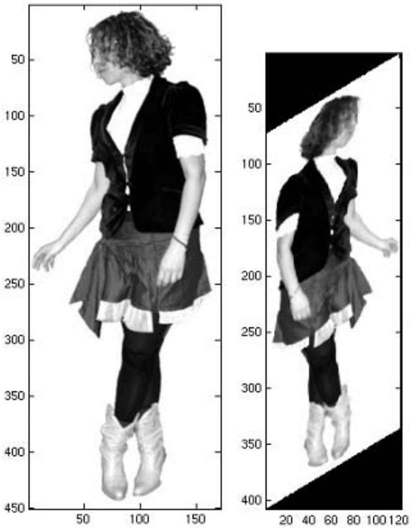


Figure H6.2 – *Applying the transformation*

```
%===== RECTTRANSF.M
% Transformation of a rectangle
fmt='jpeg'; fn='christine.jpg'; pixc=imread(fn,fmt);
figure(1); imagesc(pixc); set(gcf,'color',[1 1 1])
tbcolor=[0:1/255:1]'*[1 1 1]; colormap(tbcolor);
set(gca,'DataAspectRatio',[1 1 1],'units','pixels')
Spix=size(pixc); Nl=Spix(1); Nc=Spix(2);
xor=40; yor=40;
set(gca,'Position',[xor yor Nc Nl],'NextPlot','add');
xy=[1 1;Nc 1;Nc Nl;1 Nl]; XY=[];
plot([xy(:,1);xy(1,1)],[xy(:,2);xy(1,2)],'r')
for k=1:4
    pt=ginput(1); XY=[XY;pt]; plot(pt(1),pt(2),'oy')
    text(pt(1),pt(2),int2str(k))
end
plot([XY(:,1);XY(1,1)],[XY(:,2);XY(1,2)],'y')
set(gca,'Position',[xor yor Nc Nl],'NextPlot','replace');
%===== Calculating the transformation matrix
Mt=[];
for k=1:4
 l1=[xy(k,:) 1 zeros(1,3) -xy(k,1)*XY(k,1) -xy(k,2)*XY(k,1)];
 l2=[zeros(1,3) xy(k,:) 1 -xy(k,1)*XY(k,2) -xy(k,2)*XY(k,2)];
 Mt=[Mt;l1;l2];
end
XX=zeros(8,1); XX(:)=XY'; cof=Mt\XX;
coeff=zeros(3,3); coeff(:)=[cof;1]; coeff=coeff';
```

```
%===== Application
xc=[1:Nc]; yc=[1:Nl]';
tbx=ones(Nl,1)*xc; tby=yc*ones(1,Nc);
tbxy1=[reshape(tbx,1,Nl*Nc);reshape(tby,1,Nl*Nc);ones(1,Nl*Nc)];
idtb=(tbxy1(1,:)-1)*Nl+tbxy1(2,:); % Linear index
tbv=coeff*tbxy1;
tbv(1,:)=round(tbv(1,:)./tbv(3,:));
tbv(2,:)=round(tbv(2,:)./tbv(3,:));
xmin=min(tbv(1,:)); xmax=max(tbv(1,:)); ncol=xmax-xmin+1;
ymin=min(tbv(2,:)); ymax=max(tbv(2,:)); nlig=ymax-ymin+1;
tbidxR=[tbv(1,:)-xmin+1;tbv(2,:)-ymin+1];
pixcR=255*ones(nlig,ncol);
idtbR=(tbidxR(1,:)-1)*nlig+tbidxR(2,:);
pixcR(idtbR)=pixc(idtb);
%===== Displaying the result
figure(2); imagesc(pixcR); set(gcf,'color',[1 1 1])
colormap(tbcolor); set(gca,'DataAspectRatio',[1 1 1])
set(gca,'units','pixels','Position',[xor yor ncol nlig]);
```

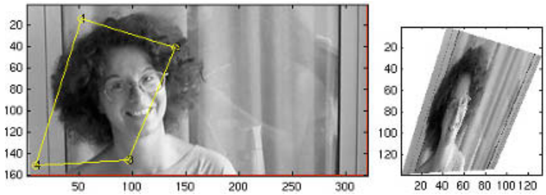


Figure H6.3 – *Applying the torsion*

H6.4 (Rectangular filter) (see page 210)

```
%===== LENARECT.M
load lena; dims=size(pixc);
figure(1); imagesc(pixc+1);
colormap(cmap); axis('image')
%===== Rectangular filter
h=ones(5,1) * ones(1,5);
h=h/25; pixr=round(filter2(h,pixc));
figure(2); imagesc(pixr+1);
colormap(cmap); axis('image')
resul=zeros(dims(1)+2,dims(2)+2); resul2=resul;
%===== Implementing the filtering with two filters
pixc==[pixc ones(dims(1),2); ones(2,dims(2)+2)];
for l=1:dims(2)+2,
    resul(:,l)=filter(ones(5,1)/5,1,pixc(:,l));
end
```

```
for k=1:dims(1)+2,
    resul2(k,:)=filter(ones(1,5)/5,1,resul(k,:));
end
%=====
resul=round(resul2(3:dims(1)+2, 3:dims(2)+2));
figure(3); imagesc(resul+1); colormap(cmap); axis('image')
```

H6.5 (Conical filter) (see page 210)

```
%===== LENACONE.M
%===== Loading the original image
load lena; figure(1); imagesc(pixc+1);
colormap(cmap); axis('image')
%===== Conical filter
h=[0= 0 1 0 0;0 2 2 2 0;...
   1 2 5 2 1;0 2 2 2 0;0 0 1 0 0];
h=h/sum(sum(h)); pixr=round(filter2(h,pixc));
figure(2); colormap(cmap);
imagesc(pixr+1); axis('image')
```

H6.6 (Gaussian smoothing filter) (see page 211)

1. Function:

```
function hg=moygauss(sigma)
%%==============================%
%%  Gaussian filter             %
%% SYNOPSIS: hg=MOYGAUSS(sigma) %
%%   sigma = standard deviation %
%%   hg    = Gaussian filter    %
%%==============================%
xx=-sigma*5:sigma*5;
g=exp(-xx.*xx / (2*sigma*sigma)); % Gaussian
%===== Keeping only the significant values
hg=g(g>max(g)*.005); hg=hg/sum(hg(:));
hg=hg' * hg;
return
```

2. Applying it to the test image (Figure H6.4):

```
%===== TSTMOYGAUSS.M
load lena; subplot(222); imagesc(pixc+1);
colormap(cmap); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
%===== Gaussian filter with sigma=4
h=moygauss(4); subplot(221); imagesc(h);
axis('image'); set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
pixr=round(filter2(h,pixc));
subplot(223); colormap(cmap); imagesc(pixr); axis('image')
```

```
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
%===== Gaussian filter with sigma=2
h=moygauss(2); pixr=round(filter2(h,pixc));
subplot(224); colormap(cmap); imagesc(pixr); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
set(gcf,'Color',[1 1 1])
```

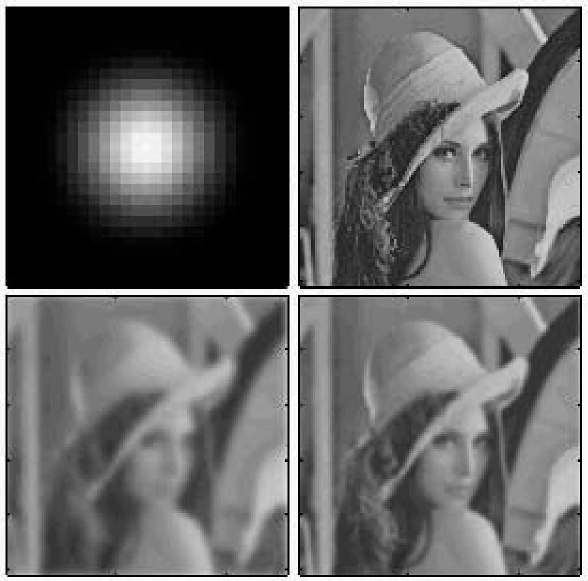


Figure H6.4 – *Effect of the Gaussian smoothing filter for two values of σ*

H6.7 (Sobel derivative filter) (see page 213)

1. The Sobel filter:

```
%===== LENASOBEL.M
load lena
set(gcf,'color',[1 1 1])
Spix=size(pixc); Nl=Spix(1); Nc=Spix(2);
colormap(cmap); NbLevel=size(cmap,1);
subplot(131); imagesc(pixc); axis('image')
%===== Sobel filter
hx=[1 0 -1;2 0 -2;1 0 -1]/4; hy=hx';
%===== x-filtering
pixrx=filter2(hx,pixc);
subplot(132); imagesc(pixrx); axis('image')
%===== y-filtering
```

```
pixry=filter2(hy,pixc);
subplot(133); imagesc(pixry); axis('image')
```

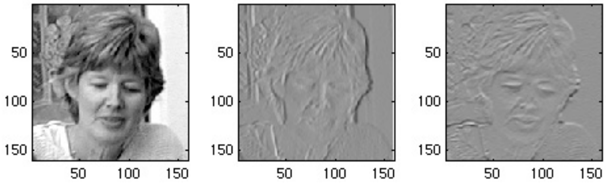


Figure H6.5 – *Differentiation with respect to the x and y axes*

2. Second derivative filter (Figure H6.6):

```
%===== LENADERSEC.M
load lena
subplot(121); imagesc(pixc);
colormap(cmap); axis('image')
%===== Second derivative filter
h=[0 1 0;1 -4 1;0 1 0];
pixr=(round(filter2(h,pixc)));
subplot(122); imagesc(pixr); axis('image');
```

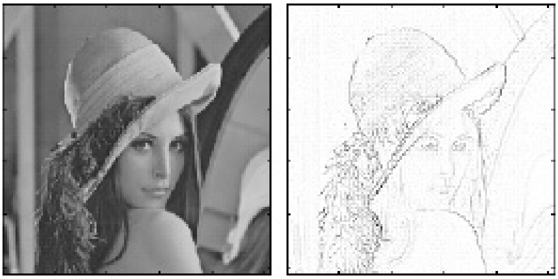


Figure H6.6 – *Second derivative*

3. Derivative filter design (Figure H6.7):

```
%===== DERIVSYNTH.M
load lena; subplot(221); imagesc(pixc); axis('image');
colormap(cmap);
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0],...
    'XTick',[],'YTick',[])
%===== Derivative filter
N=5; hx=cos(pi*(1:N)) ./ (1:N); hx=[-hx(N:-1:1) 0 hx];
```

```
hx=hx .* (.54 - .46*cos(2*pi*(0:2*N)/(2*N)));
hdp=hx' * hx;    % Derivation dans les deux directions
pixr=filter2(hdp,pixc);
%=====
subplot(223); imagesc(pixr); axis('image');
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0],...
    'XTick',[],'YTick',[])
%===== Second derivative filter
N=3; hx=2*cos(pi*(1:N)) ./ (1:N) ./ (1:N);
hx=[hx(N:-1:1) pi*pi/3 hx];
hx=hx .* (.54 - .46*cos(2*pi*(0:2*N)/(2*N)));
hds=hx' * hx;    % Derivation along both axes
pixr=filter2(hds,pixc);
%=====
subplot(224); imagesc(pixr); axis('image');
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0],...
    'XTick',[],'YTick',[])
set(gcf,'Color',[1 1 1])
```

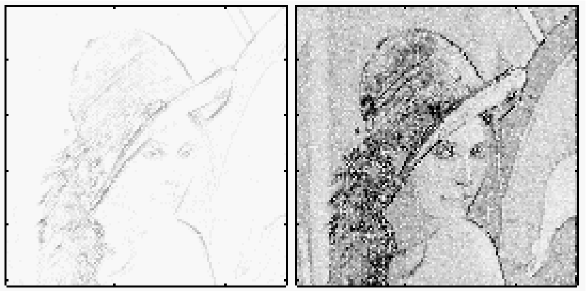


Figure H6.7 – *Derivative filter design*

H6.8 (Gaussian derivative-smoothing filter) (see page 216)

1. Type:

```
function hd=dermoygauss(n1,n2,sigma1,sigma2,theta)
%%=======================================================%
%% Gaussian derivative-smoothing filter                  %
%% SYNOPSIS: hd = DERMOYGAUSS(n1,n2,sigma1,sigma2,theta) %
%%    n1,n2 = Filter dimensions                          %
%%   sigma1 = Standard deviation in the x direction      %
%%   sigma2 = Standard deviation in the y direction      %
%%    theta = Rotation angle                             %
%%       hd = Filter PSF                                 %
%%=======================================================%
```

```
den1=sigma1*sqrt(2*pi); den2=sigma2^3*sqrt(2*pi);
s12=2*sigma1^2; s22=2*sigma2^2;
ct=cos(theta); st=sin(theta);
m1=(n1+1)/2; m2=(n2+1)/2; hd=zeros(n1,n2);
for k=1:n1
    xc=k-m1;
    for ell=1:n2
        yc=ell-m2; u=xc*ct-yc*st;
        v=xc*st+yc*ct;
        h1=exp(-u^2/s12)/den1; h2=-v*exp(-v^2/s22)/den2;
        hd(k,ell)=h1*h2;
    end
end
hd=hd / sqrt(sum(sum(abs(hd).*abs(hd))));
return
```

2. Applying the filter to the test image (Figure H6.8):

```
%===== TSTDERMOYGAUSS.M
clear; load lena; colormap(cmap);
subplot(223); imagesc(pixc); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0],'XTick',...
    [],'YTick',[])
%===== Gaussian derivative filter along y
hd=dermoygauss(20,20,3,2,pi/4);
subplot(221); mesh(hd); view(-30,20);
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
%=====
subplot(222); imagesc(1-hd); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0],'XTick',...
    [],'YTick',[])
%===== Filtering
pixr=round(filter2(hd,pixc));
subplot(224); imagesc(pixr); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0],'XTick',...
    [],'YTick',[])
set(gcf,'Color',[1 1 1])
```

H6.9 (Contours using Sobel filtering) (see page 223)

```
%===== CONTOURSOBEL.M
load lena;
subplot(121); colormap(cmap); NbLevel=size(cmap,1);
imagesc(pixc); axis('image')
%=====Sobel filter
hx=[1 0 -1;2 0 -2;1 0 -1]/4; hy=hx';
%===== x- and y-filtering
```

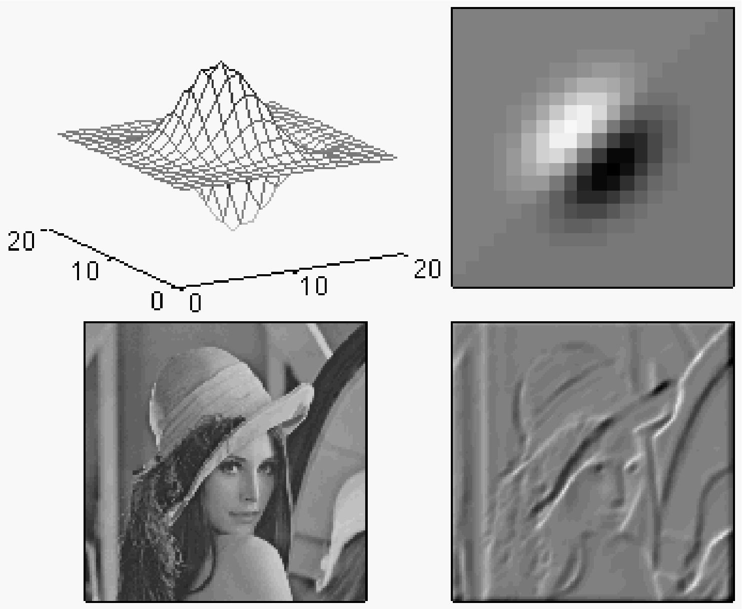


Figure H6.8 – *Applying the filter to the Gaussian derivative-smoothing filter*

```
pixrx=filter2(hx,pixc); pixry=filter2(hy,pixc);
%===== Processing the contours
pixr=sqrt(pixrx.*pixrx+pixry.*pixry);
spx=size(pixr); pixres=ones(spx);
threshold=25; [idl]=find(pixr>threshold);
pixres(idl)=NbLevel*ones(size(idl));
subplot(122); imagesc(pixres); axis('image')
```

H6.10 (Iris search) (see page 223)

A convolution is performed using the circle obtained with the function:

```
function hrnd=circpsf(radius,cmap)
%%=========================================%
%% Circular PSF                            %
%% SYNOPSIS: hrnd=CIRCPSF(radius,cmap)     %
%%    radius = radius of the circle (pixels) %
%%    cmap   = palette                     %
%%    hrnd   = N*N array                   %
%%=========================================%
rr = round(2*radius)/2 -.5;
N=rr*2+1; hrnd=zeros(N); rrn=(N-1)/2/N;
delta=1/N/sqrt(2);
cx=1/2; cy=cx; quadrx=[0.5:1:N-0.5]/N;
```

```
[X,Y]=meshgrid(quadrx); X0=X-cx; Y0=Y-cy;
x2py2=X0.*X0+Y0.*Y0; dist=abs(sqrt(x2py2)-rrn);
indc=find(dist<delta);
hrnd(indc)=ones(length(indc),1)*(size(cmap,1)-1);
return
```

The following program returns, for a sequence of radius values, the location of the circle that best corresponds to the contour of the iris.

```
%===== TRTEYES2.M
clear; load eyetst
Mcmap=size(cmap,1)-1; max0=max(max(tbmax));
tbnor=tbmax; ip0=find(tbnor>40);
tbnor(ip0)=max0*ones(1,length(ip0));
subplot(131); imagesc(tbmax); axis('image')
subplot(132); imagesc(tbnor); axis('image')
subplot(133)
seuil=230; pp=2;
for kn=1:12
    radius=11+kn/2, hrnd=circpsf(radius,cmap);
    resul=filter2(hrnd,tbnor);
    resul=normim(resul,cmap);
    [k l]=find(resul>seuil); Lk=length(k);
    for m=1:Lk,
        resul(k(m)-pp:k(m)+pp,l(m))=Mcmap*ones(2*pp+1,1);
        resul(k(m),l(m)-pp:l(m)+pp)=Mcmap*ones(1,2*pp+1);
    end
    imagesc(resul); colormap(cmap); axis('image')
    pause
end
```

`trteyes2.m` uses the function `normim`:

```
function pixn=normim(pixc,cmap)
%%==================================%
%% SYNOPSIS: pixn=NORMIM(pixc, cmap) %
%%    pixc = image                  %
%%    cmap = palette                %
%%    pixn = normalized image       %
%%==================================%
if nargin<2,
    sprintf('Erreur sur les arguments');
    return
end
plc=size(cmap,1)-1;
maxp=max(max(pixc)); minp=min(min(pixc));
pixn = round((pixc-minp) * plc/(maxp-minp));
return
```

This method, however, is not very robust as it highly dependent on the values obtained for the extracted contours.

H6.11 (Median filtering) (see page 227)
Type:

```
function imageO=median2D(imageI,M,N)
%%============================================%
%% Median filter                              %
%% SYNOPSIS: imageO=MEDIAN2D(imageI,M,N)      %
%%   imageI = image to be filtered            %
%%   M      = x dimension (odd) of the window %
%%   N      = y dimension (odd) of the window %
%%   imageO = filtered image                  %
%%============================================%
Ms2=(M-1)/2; Ns2=(N-1)/2;
MN=M*N; med=(MN+1)/2;
dims=size(imageI);
imageO = zeros(dims);
for k=(Ms2+1):(dims(1)-Ms2),
    for l=(Ns2+1):(dims(2)-Ns2),
        pixbloc=imageI(k-Ms2:k+Ms2,l-Ns2:l+Ns2);
        pixt=sort(pixbloc(:));
        imageO(k,l)=pixt(med);    % Median value
    end
end
return
```

The following program uses the `median2D` function as well as the `moygauss.m` function (created in exercise 6.6) on the “snowy” test image.

Notice in Figure H6.9 that the median filtering is better at preserving the contours and eliminating the snow than the Gaussian-smoothing filtering:

```
%===== LENAMEDIAN.M
load lena; dims=size(pixc);
subplot(221); imagesc(pixc);
colormap(cmap); axis('image')
%===== Noise (snow)
snow=(randn(dims)>-2); pixcsnow=pixc .* snow;
%===== Median filtering
pixfilmed = median2D(pixcsnow,5,5);
%===== Gaussian filter
hd = moygauss(2);              % Try 2 then 4
pixfilgauss=filter2(hd,pixcsnow);
subplot(222); imagesc(pixcsnow); axis('image')
subplot(223); imagesc(pixfilmed); axis('image')
subplot(224); imagesc(pixfilgauss); axis('image')
```

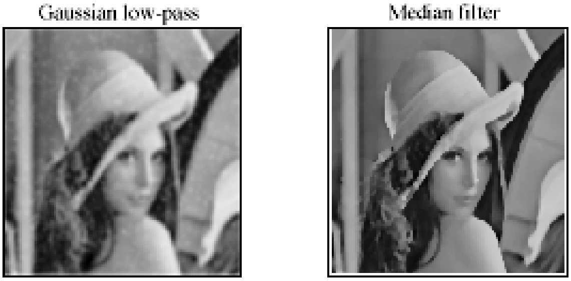


Figure H6.9 – *Suppression of the snow by gaussian-smoothing filtering (image on the left) and by median filtering (image on the right)*

H6.12 (Processing the result of a rotation) (see page 227)

1. Type:

```
%===== GEOMPROC.M
% Processing of the rotated image
pixc=imread('imageGGR.bmp','bmp');
imageS = median2D(pixc,3,3);
tbcolor=[0:1/255:1]'*[1 1 1]; % Gray colormap
imagesc(imageS); colormap(tbcolor)
set(gca,'DataAspectRatio',[1 1 1]);
imwrite(imageS,tbcolor,'imageGGRn.bmp','bmp')
```

2. The mean is calculated over the four adjacent pixels (left, right, above and below). Type:

```
%===== GEOMPROC2.M
% Processing of the rotated image
%===== Transformed image, original rotation angle
% and dimension
load pixcR2; % Image, angle and dims of the original image
spix=size(pixcR2); nl=spix(1); nc=spix(2);
imagesc(pixcR2); colormap(tbcolor)
set(gca,'DataAspectRatio',[1 1 1]);
tbx=ones(nl,1)*[1:nc]; tby=[1:nl]'*ones(1,nc);
numR=reshape(tby,1,nl*nc);
numC=reshape(tbx,1,nl*nc);
%===== Testing the pixels of the result
nbool=tstinrect(numR,numC,thet,Nl,Nc,nl,nc);
idxrect=find(nbool); Lir=length(idxrect);
idxm1=find(pixcR2(idxrect)==-1); Lim1=length(idxm1);
pp=pixcR2;
%===== Processing
```

```
for k=1:Lim1
    nn=0; sp=0; ptc=idxrect(idxm1(k));
    kkl=ptc-nl; kkr=ptc+nl;
    if kkl>=1 & pp(kkl)~-1, sp=sp+pp(kkl); nn=nn+1; end
    if kkr<=Lir & pp(kkr)~-1, sp=sp+pp(kkr); nn=nn+1; end
    kku=ptc-1; kkd=ptc+1;
    if mod(kku,nl)~=1 & pp(kku)~-1,
       sp=sp+pp(kku); nn=nn+1;
    end
    if mod(kkd,nl)~=0 & pp(kkd)~-1,
       sp=sp+pp(kkd); nn=nn+1;
    end
    pp(idxrect(idxm1(k)))=sp/nn;
end
figure(3); imagesc(pp); colormap(tbcolor);
set(gca,'DataAspectRatio',[1 1 1]);
```

The previous program uses a function (`tstinrect`) that makes it possible to detect the points of the resulting rectangle that belong to the original rectangle. Notice that we can end up with more points than in the original rectangle (Figure H6.10). The overlapping of the processed pixels is not carried out:

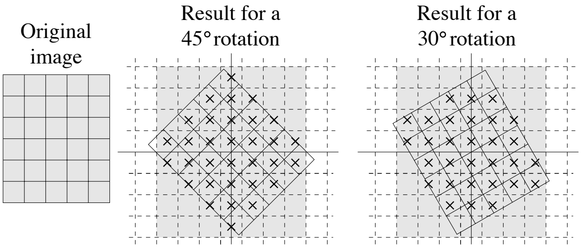


Figure H6.10 – *Effects of the rotation: the '×'s indicate the pixels of the resulting rectangle that were found to have belonged to the original rectangle subjected to the rotation*

```
function nbool=tstinrect(numR,numC,thet,Nl,Nc,nl,nc)
%%=====================================================%
%% SYNOPSIS: nbool=TSTINRECT(numR,numC,thet,Nl,Nc,nl,nc) %
%%  numR,numC = Points to be tested (vectors of the row  %
%%              and column numbers)                       %
%%  thet      = Rotation angle                            %
%%  Nl,Nc     = Number of rows and columns of the         %
%%              original image                            %
%%  nl,nc     = Number of rows and columns of the         %
%%              tranformed image                          %
%%  nbool     = Result of the test                        %
```

```
%%=======================================================%
nx=length(numR); numR=reshape(numR,1,nx);
numC=reshape(numC,1,nx);
xor=(1+nc)/2; yor=(1+nl)/2;  Xor=(1+Nc)/2;;Yor=(1+Nl)/2;
MRotM=[cos(thet) sin(thet);-sin(thet) cos(thet)];
V=[numC-xor;yor-numR];
VmR=MRotM*V;          % Reverse rotation
C1=(VmR(1,:)<Xor-0.5) & (VmR(1,:)>-Xor+0.5);
C2=(VmR(2,:)<Yor-0.5) & (VmR(2,:)>-Yor+0.5);
nbool=C1 & C2;
return
```

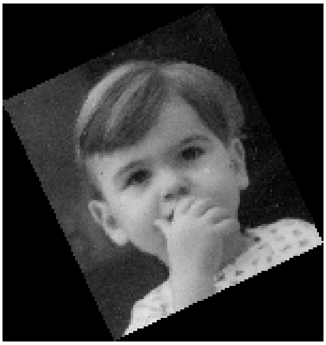


Figure H6.11 – *Result of the mean calculation operation on the transformed image*

H6.13 (Application of the Otsu method) (see page 233)

1. Threshold calculation function:

```
function [threshold,Hs]=otsu(pixc)
%%=====================================================%
%% Binarization with the Otsu method                   %
%% SYNOPSIS: [threshold,Hs]=OTSU(pixc)                 %
%%    pixc      = Image with 256 levels of gray        %
%%    threshold = Calculated optimal threshold         %
%%    Hs        = Criterium to be maximized for s=0:255 %
%%=====================================================%
%===== Histogram of the image
[nlig ncol]=size(pixc);
Lhist=256; histog=zeros(Lhist,1);
for k=1:Lhist
    histog(k)=length(find(pixc==k-1));
end
histog=histog/nlig/ncol;
%===== Calculating the criterium
Pinf=0; Psup=1; sinf=0; muinf=0; ssup=(0:Lhist-1)*histog;
```

```
%===== Calculating the criterium for the values of S
% Hs = Pinf*Psup*(muinf-musup)*(muinf-musup)
Hs=zeros(1,Lhist);
for S=0:Lhist-2
    %===== Distributions
    Pinf=Pinf+histog(S+1); Psup=1-Pinf;
    %===== Local means
    sinf=sinf+S*histog(S+1); ssup=ssup-S*histog(S+1);
    muinf=sinf/Pinf; musup=ssup/Psup;
    Hs(S+1)=Pinf*Psup*(muinf-musup)*(muinf-musup);
end
threshold=find(Hs==max(Hs)); % Threshold
return
```

2. Use of the previous function (Figure H6.12):

```
%===== BINAROTSU.M
load('elido2.mat');
nlig=size(pixc,1); ncol=size(pixc,2);
figure(1); colormap(cmap);
subplot(121); imagesc(pixc); axis('image')
%=====
thresholdOtsu=otsu(pixc);
pixc2=zeros(nlig,ncol); pixc2=255*(pixc>thresholdOtsu);
subplot(122); imagesc(pixc2); axis('image'); colormap(cmap)
```

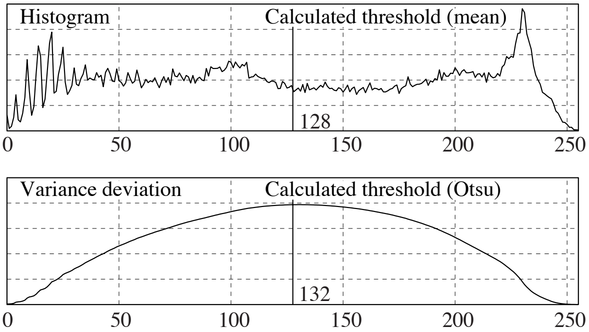


Figure H6.12 – *Results of the threshold calculation*

H6.14 (Writing basic functions) (see page 238)

1. The DCT given by expression 6.33 can be written:

$$\begin{aligned} P(u,v) &= \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} p(x,y)\cos\frac{(2x+1)\pi u}{16}\cos\frac{(2y+1)\pi v}{16} \\ &= \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\cos\frac{(2x+1)\pi u}{16}\sum_{y=0}^{7} p(x,y)\cos\frac{(2y+1)\pi v}{16} \end{aligned}$$

with $C(0) = \frac{1}{\sqrt{2}}$ and $C(k) = 1$ for $k = 1\dots7$.

The second sum corresponds to the matrix product:

$$q(x,v) = [p(x,y)] \times [ct(y,v)] \text{ with } ct(y,v) = \cos\frac{(2y+1)\pi v}{16}$$

The first sum corresponds to:

$$P(u,v) = [cs(u,x)] \times [q(x,v)] \text{ with } cs(u,x) = \cos\frac{(2x+1)\pi u}{16}$$

The matrix $N = [C(u)C(v)]/4$ is a weighting matrix, and we have:

$$P(u,v) = N.*[cs(u,x)] * [p(x,y)] * [ct(y,v)]$$

where ".*" denotes term-by-term multiplication:

```
function coeff=DCTp(pixc)
%%====================================================%
%% Calculating the DCT coefficients of an (8*8) block %
%% SYNOPSIS: coeff=DCTP(pixc)                         %
%%    pixc  = (8*8) block                             %
%%    coeff = DCT coefficients (8*8)                  %
%%====================================================%
%%    Uses the global variables mNORM, mYV, mUX:      %
%%    mNORM = [1/2 ones(1,7)/sqrt(2); ...             %
%%             ones(7,1)/sqrt(2) ones(7,7)]/4;        %
%%    mYV   = cos((2*[0:7]'+1)*[0:7]*pi/16);          %
%%    mUX   = cos([0:7]'*(2*[0:7]+1)*pi/16);          %
%%====================================================%
global mNORM mYV mUX
coeff = mNORM .*  (mUX * pixc * mYV);
return
```

2. Quantization function:

```
function matq=quant(dctcof, Qtab)
%%==================================================%
%% Quantization and rounding of the DCT coefficient %
%% SYNOPSIS: matq=QUANT(dctcof, Qtab)               %
%%    dctcof = DCT coefficients                     %
%%    Qtab   = weighting matrix                     %
%%==================================================%
matq = round(dctcof ./ Qtab);
return
```

3. Verification:

```
%===== CORRIG1.M
clear
global mNORM mYV mUX
[Qtab, zig, zag]=initctes;
pix=[139 144 149 153 155 155 155 155;
     144 151 153 156 159 156 156 156;
     150 155 160 163 158 156 156 156;
     159 161 162 160 160 159 159 159;
     159 160 161 162 162 155 155 155;
     161 161 161 161 160 157 157 157;
     162 162 161 163 162 157 157 157;
     162 162 161 161 163 158 158 158];
coeff=DCTp(pix-128);
coeffQ=quant(coeff, Qtab);
valmoy=0;
tramec=code1(coeffQ,valmoy,zag)
```

where the `code1.m` function given below creates a frame without any entropic coding:

```
function [framec,valm]=code1(coeffQ,meanval,zig)
%%=======================================================%
%% Calculating a frame without entropic coding           %
%% SYNOPSIS: [framec,valm]=CODE1(coeffQ,meanval,zig)     %
%%    coeffQ  = quantized DCT coefficients               %
%%    meanval = mean value of the previous block         %
%%    zig     = zigzag indices for reading               %
%%    framec  = coded frame  (bits numbers are set to 0) %
%%    valm    = mean used for the next block's coding    %
%%=======================================================%
bb=0;
if (coeffQ(1,1)==0), coeffQ(1,1)=eps; bb=1; end
frame=coeffQ(zig);         % coeffQ read in zigzag
%===== Indices of the non-zero terms
idz=find(frame); lidz=length(idz);
```

```
nbzer=diff(idz)-1; lnbz=length(nbzer);
valm=coeffQ(1,1); framec=[17 valm-meanval];
for ik=1:lnbz
    framec=[framec nbzer(ik) 17 frame(idz(ik+1))];
end
framec=[framec 0 0];
if bb, framec(2)=-meanval; end
return
```

H6.15 (Writing the compressed frame) (see page 239)

1.

```
%===== UNBLOCCODE.M
clear
global mNORM mYV mUX
[Qtab, zig, zag]=initctes;
%===== The test image
pix=[139 144 149 153 155 155 155 155;
     144 151 153 156 159 156 156 156;
     150 155 160 163 158 156 156 156;
     159 161 162 160 160 159 159 159;
     159 160 161 162 162 155 155 155;
     161 161 161 161 160 157 157 157;
     162 162 161 163 162 157 157 157;
     162 162 161 161 163 158 158 158];
pixc=[pix pix pix ones(8,8)*255];
[nl,nc]=size(pixc);
%===== Analysis and coding of the block
nbx=floor(nc/8); nby=floor(nl/8);
fid=fopen('unbloccode.dat','w');
fwrite(fid,nby,'int8'); fwrite(fid,nbx,'int8');
nbbits=0; meanval=0;
for indy=1:nby
    for indx=1:nbx
        idx=indx*8; idy=indy*8;
        pix=pixc(idy-7:idy,idx-7:idx);
        cofpix=DCTp(pix-128); coeffQ=quant(cofpix, Qtab);
        [tramec,valm]=code1(coeffQ,meanval,zag);
        fwrite(fid,tramec,'int8');
        meanval=valm;
    end
end
fclose(fid);
```

2.

```
%===== IMGTSTCODE.M
clear
global mNORM mYV mUX
[Qtab, zig, zag]=initctes;
load wendyg
figure(1); imagesc(pixc); colormap(cmap); axis('image')
[nl,nc]=size(pixc);
```

```
%===== Analysis and coding of each block
nbx=floor(nc/8); nby=floor(nl/8);
fid=fopen('imgtstcode.mat','w');  % Coefficients
fwrite(fid,nby,'int8');           % Header, number of rows
fwrite(fid,nbx,'int8');           % number of columns
nbbits=0; meanval=0;
for indy=1:nby
    idy=indy*8;
    for indx=1:nbx
        idx=indx*8; pix=pixc(idy-7:idy,idx-7:idx);
        cofpix=DCTp(pix-128); coeffQ=quant(cofpix, Qtab);
        [framec,valm]=code1(coeffQ,meanval,zig);
        fwrite(fid,framec,'int8');    % Writing the frame
        meanval=valm;
    end
end
fclose(fid);
return
```

H6.16 (Decompression) (see page 240)

1.

```
function  dctcof=unquant(matq, Qtab)
%%=========================================%
%% Dequantization of the DCT coefficients %
%% SYNOPSIS: dctcof=UNQUANT(matq, Qtab)   %
%%    Qtab = Quantization matrix          %
%%    matq = Quantized DCT coefficients   %
%%=========================================%
dctcof = matq .* Qtab;
return
```

2.

```
function pixels=iDCTp(Pdct)
%%=============================================%
%% Calculating the inverse DCT                 %
%% SYNOPSIS: pixels=IDCTP(Pdct)                %
%%    Pdct   = DCT coefficients (8*8)          %
%%    pixels = (8*8) block                     %
%%=============================================%
%% Uses the global variables mNORM, mYV, mUX. %
%%=============================================%
global mNORM mYV mUX
pixels = mYV * (mNORM .* Pdct) * mUX;
return
```

3. Test sequence obtained for the decoding:

```
framec =
    17    15     1    17    -2     0    17    -1     0
    17    -1     0    17    -1     2    17    -1     0
    17    -1     0     0
```

```
valm =
    15
framec =
    17     0     1    17    -2     0    17    -1     0
    17    -1     0    17    -1     2    17    -1     0
    17    -1     0     0
valm =
    15
framec =
    17     0     1    17    -2     0    17    -1     0
    17    -1     0    17    -1     2    17    -1     0
    17    -1     0     0
valm =
    15
framec =
    17    49     0     0
valm =
    64
```

4.

```
%===== IMGTSTDECODE.M
global mNORM mYV mUX
[Qtab, zig, zag]=initctes;
%===== File containing the codes
fid=fopen('imgtstcode.mat','r');
nby=fread(fid,1,'int8');      % Number of row blocks
nbx=fread(fid,1,'int8');      % Number of column blocks
framec=fread(fid,'int8');     % Data
fclose(fid);
pixcr=zeros(8*nby,8*nbx); pix0=0; idx=1;
%===== For each block...
for indy=1:nby
    id8=indy*8;
    for indx=1:nbx
        coeffQ=zeros(1,64); kz=1;
        idx=idx+1; a=framec(idx);
        a=a+pix0; coeffQ(zig(kz))=a; pix0=a;
        idx=idx+1; kz=kz+1;
        %===== Reading a frame associated to a block
        while (framec(idx)~=0 | framec(idx+1)~=0),
            if framec(idx)~=0, kz=kz+framec(idx); end
            coeffQ(zig(kz))=framec(idx+2);
            kz=kz+1; idx=idx+3;
        end
        idx=idx+2;
        coeffQ=reshape(coeffQ,8,8);
        coeff=unquant(coeffQ, Qtab);
        pix=fix(iDCTp(coeff))+128;
        pixcr(id8-7:id8,indx*8-7:indx*8)=pix;
    end
end
%===== Comparison of the images
load wendyg
```

```
subplot(121); imagesc(pixc'); axis('image'); colormap(cmap);
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
subplot(122); imagesc(pixcr'); axis('image'); colormap(cmap);
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
set(gcf,'Color',[1 1 1])
return
```

H6.17 (Yeung and Wong method) (see page 242)

1. The binary mark that was chosen is visible in Figure H6.13, in the bottom-left corner.

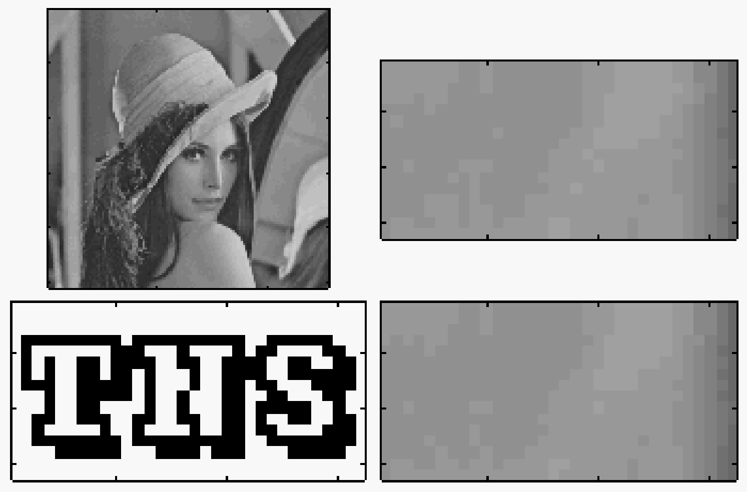


Figure H6.13 – *Inserting the mark: the image in the bottom-right corner is the part that was modified by the mark*

2. Inserting the mark:

```
%===== YEUNG.M
clear; load lena    % pixc, cmap
[nblig, nbcol]=size(pixc);
load tnslogo        % pixct, cmap
[nlt, nct]=size(pixct);
subplot(221); imagesc(pixc); colormap(cmap); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
subplot(223); imagesc(pixct); colormap(cmap); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
logo=pixct/255; % The levels are set to 0 and 1
%===== Generation of the function f
```

```
rand('seed',1);
fdef=rem(round(rand(1,256)*255),2);
%===== Top left block of the original image
bloc=pixc(1:nlt,1:nct);
subplot(222); imagesc(bloc); colormap(cmap); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
for k=1:nlt
    for l=1:nct
        ig=bloc(k,l); mg=logo(k,l);
        bloc(k,l)=modifYW(ig,mg,fdef);
    end
end
pixc(1:nlt,1:nct)=bloc;
subplot(224); imagesc(bloc); colormap(cmap); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
set(gcf,'Color',[1 1 1])
save lenat pixc cmap fdef
```

The modification implemented by the following function is designed in such a way as to have "faint" modifications of the levels of gray. A close level of grey is chosen, immediately above, or if there is none, the first level below that will satisfy the constraint:

```
function igs=modifYW(ig,mg,fdef)
%%============================================%
%% If ig~mg then ig is modified               %
%% SYNOPSIS: igs=MODIFYW(ig,mg,fdef)          %
%%    ig   = "color" of the pixel of the image %
%%    mg   = "color" of the pixel of the mark  %
%%    fdef = key function                      %
%%    igs  = modified "color"                  %
%%============================================%
ig0=ig;
while fdef(ig)~=mg
    if ig<=255, ig=ig+1; end
end
if ig==256,
   ig=ig0
   while fdef(ig)~=mg,
      if ig<=255, ig=ig-1; end
   end
end
igs=ig;
return
```

3. Reconstruction of the mark: notice that the method lacks robustness when subjected to a simple noise with a low amplitude (Figure H6.14):

```
%===== YEUNGR.M
clear; load lenat   % Watermarked image and key
[nblig, nbcol]=size(pixc);
%=====
load tnslogo        % Original mark
[nlt, nct]=size(pixct);
subplot(221); imagesc(pixc); colormap(cmap); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
%===== Top left original block
bloc=pixc(1:nlt,1:nct);
xx=fdef(bloc); resul=reshape(xx,nlt,nct);
subplot(223); imagesc(resul); colormap(cmap); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
%===== The image is noised
A=1; mnoise=round(A*rand(nblig, nbcol));
pixc=pixc+mnoise;
subplot(222); imagesc(pixc); colormap(cmap); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
bloc=pixc(1:nlt,1:nct);
xx=fdef(bloc); resul=reshape(xx,nlt,nct);
subplot(224); imagesc(resul); colormap(cmap); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
set(gcf,'Color',[1 1 1])
```

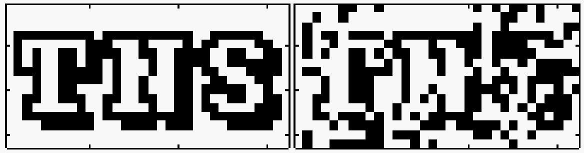


Figure H6.14 – *Condition of the mark with and without noising*

H6.18 (DCT modulation) (see page 244)

1. Calculation of the direct and inverse DCTs:

```
function coefDCT=DCTG(pixc)
%%================================================%
%% Generalized DCT function                        %
%% SYNOPSIS: coefDCT=DCTG(pixc)                    %
%%    pixc    = Array corresponding to a color plane %
%%    coefDCT = DCT coefficients                   %
%%================================================%
[Ny,Nx]=size(pixc);
alpha=sqrt(Nx*Ny)/2;
mUX = cos([0:Ny-1]'*(2*[0:Ny-1]+1)*pi/2/Ny);
```

```
mYV = cos((2*[0:Nx-1]'+1)*[0:Nx-1]*pi/2/Nx);
mNORM = [1/2 ones(1,Nx-1)/sqrt(2); ...
         ones(Ny-1,1)/sqrt(2) ones(Ny-1,Nx-1)]/alpha;
coefDCT = mNORM .*  (mUX * pixc * mYV);
return
```

```
function pixc=IDCTG(coefDCT)
%%====================================================%
%% Inverse generalized DCT function                   %
%% SYNOPSIS: coefDCT=IDCTG(pixc)                      %
%%    pixc    = Array corresponding to a color plane %
%%    coefDCT = DCT coefficients                      %
%%====================================================%
[Ny,Nx]=size(coefDCT);
alpha=sqrt(Nx*Ny)/2;
mUX = cos((2*[0:Ny-1]+1)'*[0:Ny-1]*pi/2/Ny);
mYV = cos([0:Nx-1]'*(2*[0:Nx-1]+1)*pi/2/Nx);
mNORM = [1/2 ones(1,Nx-1)/sqrt(2); ones(Ny-1,1)/sqrt(2) ...
                                     ones(Ny-1,Nx-1)]/alpha;
pixc = mUX * (mNORM .* coefDCT) * mYV;
return
```

2. Marking of the image:

```
%===== MODDCT.m
clear; load wendyg; pixc=pixc'; subplot(221)
imagesc(pixc); axis('image'); colormap(cmap)
set(gcf,'Color',[1 1 1])
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
[nlt,nct]=size(pixc);
%===== Calculating the DCT
coeff = DCTG(pixc-128); mcoeff=abs(coeff);
zzc=zeros(nlt*nct,1); zzc(:)=coeff;
zz=zeros(nlt*nct,1); zz(:)=mcoeff;
%===== Looking for the coeffts to be modified
[zzt,idzz]=sort(zz); zzt=flipud(zzt); idzz=flipud(idzz);
zzcm=zzc(idzz);
zztt=filter([1],[1 -1],zzt .* zzt);
sigT=zzt'*zzt;           % Estimation of the power
parP=0.90;               % Portion of the power
idd=find(zztt<=parP*sigT);
%===== Binary sequence length
Ls=max(idd)+1; rand('seed',3);
kcond=1; seqbin=kcond*(2*round(rand(Ls-1,1))-1);
%===== Some coefficients are modified
zzcm(2:Ls)=zzcm(2:Ls)+seqbin; zzc(idzz)=zzcm;
W=zeros(nlt*nct,1); W(2:Ls)=seqbin; W(idzz)=W;
```

```
W=reshape(W,nlt,nct);
coeffm=reshape(zzc,nlt,nct);
pixcm=iDCTG(coeffm)+128;
subplot(222); imagesc(pixcm); axis('image'); colormap(cmap)
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
subplot(223); imagesc(DCTG(W)); axis('image');
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
save wendygm pixcm cmap W
```

The program's parameter **kcond** can be used to prevent a quantization, such as the one used in the JPEG coding, from destroying the mark.

3. Checking the marking and the mark:

```
%===== DEMODDCT.M
clear; load wendygm;      % Marked image
load wendyg; pixc=pixc';  % Original Image with the mark W
subplot(221); imagesc(pixc); axis('image'); colormap(cmap)
subplot(222); imagesc(pixcm); axis('image'); colormap(cmap)
pixcd=pixcm-pixc;
subplot(224); imagesc(pixcd); axis('image'); colormap(cmap)
title('Difference')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
coeffd=DCTG(pixcd-128); coeffd(1,1)=0;
%===== The owner is authentified by comparing the marks
max(max(abs(coeffd-W)))
```

In practice, it is better to compare the original mark and the reconstructed mark using a resemblance measurement such as the correlation, in order to take into account the alterations the image may have undergone during its transmission.

H7 Random variables

H7.1 (Confidence ellipse) (see page 264)

1. By using the indicated variable change $\mathbf{Y} = \mathbf{C}^{-1/2}(\mathbf{X} - \mathbf{m})$, then by changing over to the polar coordinates, we successively get:

$$\begin{aligned}
\alpha &= \int_{\Delta(s)} \frac{1}{2\pi\sqrt{\det(\mathbf{C})}} \exp\left(\frac{1}{2}(\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})\right) dx_1 dx_2 \\
&= \int_{\{(y_1,y_2)\in\mathbb{R}^2 : y_1^2+y_2^2<s\}} \frac{1}{2\pi} \exp\left(-\frac{1}{2}(y_1^2+y_2^2)\right) dy_1 dy_2 \\
&= \frac{1}{2\pi}\int_0^{\sqrt{s}}\int_0^{2\pi} e^{-r^2/2} r dr d\theta = \int_0^{s/2} e^{-u} du = 1 - e^{-s/2}
\end{aligned}$$

hence $s = -2\log(1-\alpha)$.

2. Type:

```
%===== CELLCONF.M
N=200; alpha=0.95;
s=-2*log(1-alpha); Nth=fix((1-alpha)*N);
C=[2.3659 -0.3787;-0.3787 0.6427];
y=randn(2,N); x=sqrtm(C)*y;
ellipse([0 0],inv(C),s); hold;
plot(x(1,:),x(2,:),'+'); hold; axis([-6 6 -6 6]);
normy = ones(1,2) * (y .* y);
nb=length(find(normy>s));
text(2,4,sprintf('Nth=%g   N=%g',Nth,nb))
```

H7.2 (Poisson distribution) (see page 266)

1. The mean is given by:

$$\mathbb{E}\{X\} = \sum_{k=0}^{+\infty} e^{-a} k a^k / k! = a$$

and the variance by $\mathrm{var}(X) = \mathbb{E}\{X^2\} - \mathrm{E}^2\{X\}$. Because we have:

$$\mathbb{E}\{X(X-1)\} = \sum_{k=0}^{+\infty} e^{-a} \frac{k(k-1)}{k!} a^k = \sum_{k=0}^{+\infty} e^{-a} a^2 \frac{1}{(k-2)!} a^{k-2} = a^2$$

then $\mathbb{E}\{X^2\} = \mathbb{E}\{X(X-1)\} + \mathbb{E}\{X\} = a^2 + a$. Therefore, $\mathrm{var}(X) = a$.

2. $F_X(k) = P_r(X \le k)$ can be written $F_X(k) = F_X(k-1) + p_X(k)$ and to avoid calculating the factorial function, the following recursive form is used for calculating $p_X(k)$:

$$p_X(k) = a p_X(k-1)/k$$

3. In order to generate a Poisson variable with the parameter a, we can use the following method:

   (a) generate the sequence U by typing `U = rand(1,N)`. The maximum value is denoted by $U_{\max}$;

   (b) construct, as a function of a, the table of the cumulative probabilities $F_X(k) = P_r(X \le k)$ smaller than $U_{\max}$;

   (c) determine, for each $U(k)$, the highest integer $X(k)$ such that $F_X(X(k)) \le U(k)$. We can use `find((X>=a)&(X<b))` which extracts the indices of the values of x between a and b.

```
%===== CPOISSON.M
clear; a=5; N=2000; U=rand(1,N); X=zeros(1,N);
Umax=max(U); p(1)=exp(-a); % P(X=0)
FX(1)=p(1); nmax=1;
%===== cdf
while ((FX(nmax)<Umax)&(nmax<N))
    p(nmax+1)=a * p(nmax) / nmax; % Iterative calc.
    FX(nmax+1)=FX(nmax) + p(nmax+1);
    nmax=nmax+1;
end
%===== Generation of X
for ii=1:nmax-1
    ind=find((U>=FX(ii))&(U<FX(ii+1)));
    X(ind)=ii * ones(1,length(ind));
end
%===== Extreme values
ind=find(U<FX(1)); X(ind)=zeros(1,length(ind));
ind=find(U>=FX(nmax)); lind=length(ind);
X(ind)=nmax * ones(1,length(ind));
%===== Verification
[pest x0]=hist(X,(0:nmax)); stem(x0,pest/N)
hold; plot((0:nmax-1),p,(0:nmax-1),p,'x'); hold
```

H7.3 (Rayleigh distribution) (see page 269)

1. We get:

$$u = \int_0^x \frac{t}{\sigma^2} \exp\left(-\frac{t^2}{2\sigma^2}\right) dt = 1 - \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

   If we solve for x, we get $x = \sigma\sqrt{-2\log(1-u)}$. Hence, if U has a uniform distribution on $(0,1)$, $X = \sigma\sqrt{-2\log(1-U)}$ has a Rayleigh distribution. Because U and $1-U$ have the same distribution, it is equivalent to taking $X = \sigma\sqrt{-2\log(U)}$ instead.

2. Type the program:

```
%===== CRAYLEIGH.M
clear; N=3000;
U=rand(1,N); X=sqrt(-2*log(U));
lk=(max(X)-min(X))/20; maxx=max(X);
pointsx=(0:lk:maxx);
[nn xx]=hist(X,pointsx);
%===== Estimating the pdf
pest=nn/(N*lk); bar(pointsx,pest)
%===== Theoretical Rayleigh pdf
pth=pointsx .* exp(-pointsx .* pointsx /2);
hold on; plot(pointsx,pth,'or'); hold off; grid
```

Figure H7.1 shows the theoretical graph for the probability density of a Rayleigh distribution and the histogram of the values obtained by the generator.

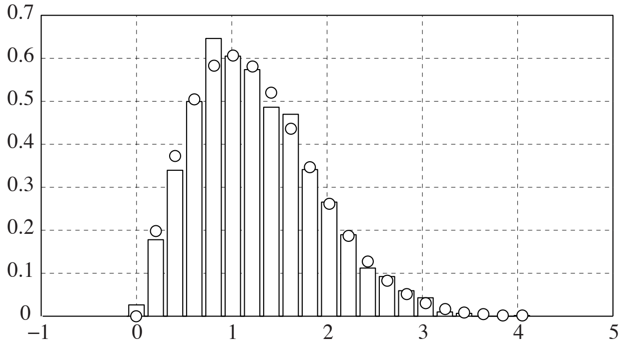


Figure H7.1 – *Rayleigh distribution: probability density* (∘) *and histogram of the simulated length* 3000 *sample*

H7.4 (Bernoulli distribution) (see page 269)

1. $\mathbb{E}\{B_n\} = 1 \times \Pr(B_n = 1) + 0 \times \Pr(B_n = 0) = p$. Likewise, $\mathbb{E}\{B_n^2\} = p$ and therefore $\text{var}(B_n) = p - p^2 = p(1-p)$.

2. Because the random variables are assumed to be independent, they are necessarily uncorrelated, and we have, for $k \neq n$, $\mathbb{E}\{B_k B_n\} = \mathbb{E}\{B_k\}\mathbb{E}\{B_n\} = p^2$.

3. Because of linearity, $\mathbb{E}\{S\} = \mathbb{E}\{B_n\} = p$. In order to determine the variance, we first have to calculate the second order moment $\mathbb{E}\{S^2\}$:

$$\begin{aligned}
\mathbb{E}\{S^2\} &= \frac{1}{N^2}\sum_{n=1}^{N}\sum_{k=1}^{N}\mathbb{E}\{B_n B_k\} \\
&= \frac{1}{N^2}\sum_{n=1}^{N}\mathbb{E}\{B_n^2\} + \frac{1}{N^2}\sum_{n\neq k}\mathbb{E}\{B_n B_k\} \\
&= \frac{1}{N}p + \frac{N(N-1)}{N^2}p^2 = p^2 + \frac{p(1-p)}{N}
\end{aligned}$$

In the end, $m = \mathbb{E}\{S\} = p$ and $\sigma^2 = \text{var}(S) = \mathbb{E}\{S^2\} - \mathrm{E}^2\{S\} = pq/N$.

4. We have successively $\epsilon_r^2 = 4\sigma^2/m^2 = 4(1-p)/Np \simeq 4/Np$ if we assume that $p \ll 1$, meaning that $N \approx 4/(p\epsilon_r^2)$. For $\epsilon_r = 0.1$, we get $N = 400/p$.

5. The inequalities 7.26 show that, in order to determine the random variable B that has a Bernoulli distribution with a parameter p, from a random variable U uniformly distributed on $(0, 1)$, we have to choose $B = 0$ if $0 \leq U < (1 - p)$ and 1 otherwise. This result is obtained in the following program with the instruction `B=(U>q)` where $q = 1 - p$:

```
%===== CBERNOU.M
p=0.1; q=1-p;
er=.1; N=fix(q/(p*er*er));
U=rand(1,N); B=(U>q);
mean(B)
```

H7.5 (Signal-to-quantization noise ratio) (see page 272)

1. Since all the amplitudes are assumed to belong to $(-F\sigma, +F\sigma)$, the quantization step is given by $q = 2F\sigma/2^N$. Because $\mathbb{E}\left\{\varepsilon^2\right\} = q^2/12$, we have:

$$RSB = 10\log_{10}\left(\frac{12\sigma^2}{4F^2\sigma^2}2^{2N}\right) = 6N + 10\log_{10}(3/F^2)$$

Thus, we get the following practical rule: the signal-to-noise ratio is increased by 6 dB for every additional coding bit.

2. Type:

```
function [rcval,code]=bincoding(x,N,A)
%%=========================================%
%% Two's complement binary coding          %
%% SYNOPSIS: [rcval,code]=bincoding(x,N,A) %
%%   x     = Sequence to be coded          %
%%   N     = 2^N codes (default 4)         %
%%   A     = Conversion between -A and A   %
%%   rcval = Reconstructed value           %
%%   code  = Two's complement binary code  %
%%           -2^(N-1)<=code<=2^(N-1)-1     %
%%=========================================%
if nargin<3, A=1; end
if nargin<2, N=4; end
if nargin<1, error('Parameters missing...'); end
q=A/2^(N-1); code=floor(x/q);
%===== Clipping
cm=-2^(N-1); cM=2^(N-1)-1;
idm=find(code<cm); idM=find(code>cM);
code(idM)=cM*ones(size(idM));
code(idm)=cm*ones(size(idm));
rcval=code*q+q/2;
return
```

Notice the use of the **floor** function to directly calculate the reconsructed values **yt** corresponding to the values of the unclipped signal **xt**. Of course, some values of the signal **xt** are greater than A_c, and are therefore clipped. Their number must be negligible for the chosen value of F.

3. The following program displays the histogram of the error signal and both the theoretical and the measured ratios $(S/B)_{dB}$ plotted against N (Figure H7.2). The peak value $A_c = F\sqrt{P}$ is obtained from an estimation P of the signal's power and from an a priori peak factor of 3.5:

```
%===== CSSURBQ.M
clear; T=2000; xt=randn(1,T);
%===== RLF
Fc==3.5; Cc=10*log10(3/(Fc*Fc));
%===== Ac from the estimated power
px=xt*xt'/T; Ac=Fc*sqrt(px);
%===== Indices for which there is saturation
indmax=find(xt>=Ac); lindmax=length(indmax);
indmin=find(xt<=-Ac); lindmin=length(indmin);
figure(1);
%=====
for Nb=1:7
    [yt,code]=bincoding(xt,Nb,Ac);
    ninter=2^Nb;
    q=2*Ac/ninter;        %==== Quantization step
    errq=xt-yt;           %==== Quantization error
    %===== Histogram
    hist(errq,(-5:5)*q/5); grid; pause
    pe=errq*errq'/T; ssb(Nb)=10*log10(px/pe);
    %===== Theoretical values
    ssbth(Nb)=6*Nb+Cc;
end
figure(2); plot([1:7],ssbth,':y',[1:7],ssb,'Xy'); grid
```

By examining the histograms, we can check the uniform distribution of the quantization error hypothesis. As you can also see, the power of the error is close to $q^2/12$.

H8 Random processes

H8.1 (Suppressing an affine trend) (see page 290)

1. $\{B(n)\}$ is a sequence of N Gaussian, centered random variables, uncorrelated, and is therefore independent. Hence, we have:

$$p_{\mathbf{B}}(b_0, \cdots, b_{N-1}) = \frac{1}{(\sigma_b\sqrt{2\pi})^N} \exp\left(-\frac{b_0^2 + \cdots + b_{N-1}^2}{2\sigma_b^2}\right)$$

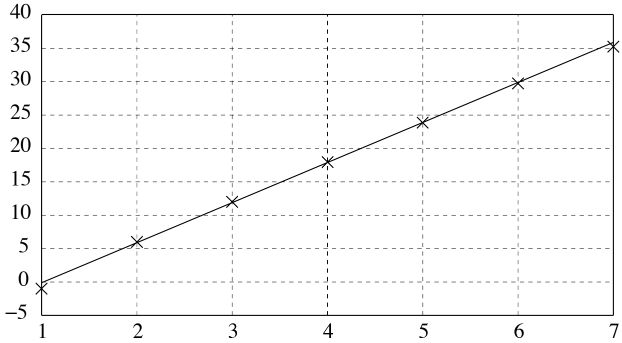


Figure H7.2 – *Both the theoretical and the measured "Signal-to-noise" ratios (SNR) of a uniform quantization plotted against the number of coding bits. The "×"s are obtained from a sample with the size 2,000*

Because $B(n) = X(n) - a_1 - a_2 n$, the probability distribution of $(X(0), \ldots, X(N-1))$ has the density:

$$p_{\mathbf{X}}(x_0, \cdots, x_{N-1}; a_1, a_2, \sigma_b^2) = \frac{1}{(\sigma_b\sqrt{2\pi})^N} \exp\left(-\frac{\sum_{n=0}^{N-1}(x_n - a_1 - a_2 n)^2}{2\sigma_b^2}\right)$$

2. Maximizing $p_{\mathbf{X}}(x_0, \ldots, x_{N-1}; a_1, a_2, \sigma_b^2)$ with respect to a_1 and a_2 is equivalent to minimizing the sum found in the exponential, that is to say $J(a_1, a_2) = \sum_{n=0}^{N-1}(x_n - a_1 - a_2 n)^2$. By differentiating $J(a_1, a_2)$ with respect to a_1 and a_2, and setting the derivatives to zero, we get the system of two equations:

$$\begin{cases} a_1 \sum_{n=0}^{N-1} 1 + a_2 \sum_{n=0}^{N-1} n = \sum_{n=0}^{N-1} x_n \\ a_1 \sum_{n=0}^{N-1} n + a_2 \sum_{n=0}^{N-1} n^2 = \sum_{n=0}^{N-1} n x_n \end{cases}$$

which can be written, with the suggested notations, $\mathbf{W}^T\mathbf{W}\mathbf{a} = \mathbf{W}^T\mathbf{X}$. Because $\mathbf{W}^T\mathbf{W}$ is invertible, the solution is $\mathbf{a} = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\mathbf{X}$, which happens to be the solution for the least squares estimator (page 393). This is not surprising since maximizing the probability density in the Gaussian case is equivalent to minimizing the square deviation.

3. The following function eliminates the affine trend:

```
function [A,dx]=tendoff(x)
%%==================================%
%% Suppression of an affine trend    %
%% SYNOPSIS: [A,dx]=TENDOFF(x)       %
%%     x  = Input sequence           %
%%     A  = Affine regression coeffts %
%%     dx = Residual                 %
%%==================================%
x=x(:); N=length(x); w=[ones(N,1) (0:N-1)'];
A=w\x; dx=x-w*A;
return
```

H8.2 (Smoothing filtering of noise) (see page 299)

1. Starting with the definition, we get:

$$H(f)=\sum_{n=-\infty}^{+\infty} h(n)e^{-2j\pi fn}=\sum_{n=0}^{7}\frac{1}{8}\,e^{-2j\pi fn}\Rightarrow |H(f)|^2=\left|\frac{\sin(8\pi f)}{8\sin(\pi f)}\right|^2$$

2. Because the input process is white with a variance of 1, $S_{XX}(f)=1$ and therefore $S_{YY}(f)=|H(f)|^2$.

3. The output power can simply be written:

$$P=\int_{-1/2}^{1/2} S_{YY}(f)df=\int_{-1/2}^{1/2}|H(f)|^2 df=\sum_{n=0}^{7}|h(n)|^2=1/8$$

4. The output autocovariance function is obtained as the inverse Fourier transform of $S_{YY}(f)$. This is also the convolution of a rectangle of width $L_h=8$ with itself, which results in the triangle with the support $(-(L_h-1), L_h-1)$. Theoretically, $R_{YY}(k)$ is therefore null for $|k|\geq L_h$.

5. The following program generates a trajectory for the signal $y(t)$ and estimates 10 points of the autocovariance function. We have to check that the estimated function is negligible beyond $L_h=8$:

```
%===== CMEANNOISE.M
T=2000; N=10;     % Number of covariance coeffts
%===== Generating the signal using a rectangular FIR
Lh=8; h=ones(1,Lh)/Lh; w=randn(1,T); x=filter(h,1,w);
%===== Estimation of the autocovariance
xi=[x zeros(1,N-1)];
D=toepl(xi,[xi(1) zeros(1,N-1)]); R=D*xi'/T;
%===== Drawing the autocovariance
tau=(-(N-1):(N-1)); Rs=[R(N:-1:2) ; R];
subplot(211); plot(tau,Rs,'o'); grid
%===== Drawing the spectrum
Lfft=512; freq=(0:Lfft-1)/Lfft;
Sf=10*log10(abs(fft(R,Lfft)));
subplot(212); plot(freq,Sf); grid
```

Figure H8.1 shows the $N = 10$ point estimate of the autocovariance function. The triangular shape is visible for $0 \leq |k| \leq 7$. Beyond that, the values are almost equal to zero, as predicted by the theoretical results.

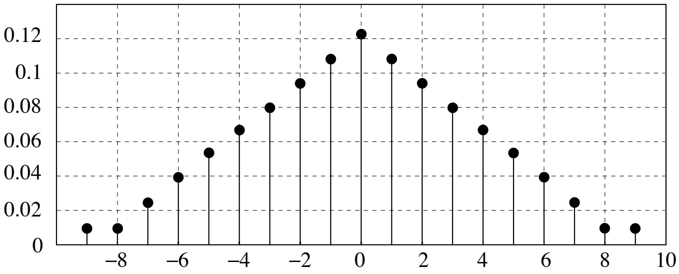


Figure H8.1 – *Triangular autocovariance function*

We have to check that the estimated value in $k = 0$, which represents the power of $Y(n)$, is in agreement with the theoretical value of $1/8$.

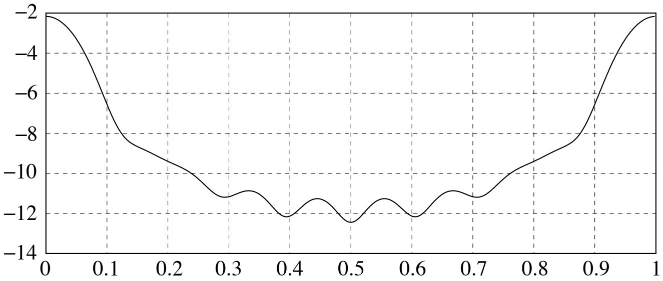


Figure H8.2 – *Spectrum estimate*

The spectrum, calculated from the FFT estimation of the autocovariance is represented in Figures H8.2 for $f \in (0, 1/2)$. It corresponds to the theoretical spectrum $|H(f)|^2$. The width of its main lobe is equal to 0.25.

H8.3 (Generating a band limited process) (see page 300)
To shift the frequencies back to the $(-1/2, +1/2)$ band, we have to divide the frequency scale expressed in Hz by the sampling frequency F_s = 10,000 Hz. Hence the spectrum band for which the process is different from zero is the $(-0.1, +0.1)$ band. Its psd is shown in Figure H8.3. Because the power is equal to the integral of the psd, $P = 0.2\alpha$ and therefore $\alpha = 10$.

The following method can be used to obtain a trajectory:

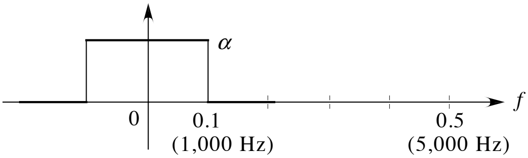


Figure H8.3 – *Power spectral density of the low-pass process*

- with the use of the **randn** function, construct a random sequence the spectral density of which is equal to 1 in the $(-1/2, +1/2)$ band;
- multiply this sequence by $\sqrt{10}$, to obtain a power equal to 10;
- apply a filter with a gain 1 to the $(-0.1, +0.1)$ band. To generate the coefficients of this filter, use the **rif** function (page 599).

Type:

```
%===== CPABLIM.M
Fs=10000; Fc=1000; fcr=Fc/Fs; T=1000;
w=sqrt(10)*randn(1,T); B=rif(50,fcr); x=filter(B,1,w);
subplot(211); plot(w); grid;
subplot(212); plot(x); grid
```

The trajectory of **x** is less "chaotic" than that of **w**. This phenomenon is analogous to the one observed for deterministic signals: reducing the frequency band leads to slower time fluctuations.

H8.4 (Pre-emphasis and de-emphasis) (see page 300)

1. If we use the filtering formula, we get the *psd* of the output signal $W(n)$ of the filter with $B(n)$ as its input. This leads to the expression of the power:

$$\mathbb{E}\{W^2(n)\} = \int_{-1/2}^{+1/2} |H_d(f)|^2 S_B(f)df$$

Because these two cascaded filters have no effect on the signal $X(n)$ (Figure H8.4), the power of the output signal's useful part is given by $\int_{-1/2}^{+1/2} S_X(f)df$.

Hence the signal-to-noise ratio has the expression:

$$\rho_{PD} = \frac{\int_{-1/2}^{+1/2} S_X(f)df}{\int_{-1/2}^{+1/2} |H_d(f)|^2 S_B(f)df}$$

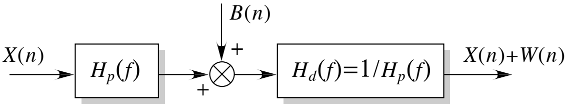


Figure H8.4 – *The pre-emphasis and de-emphasis system*

2. Using the filtering formula, we get the *psd* of the output signal of the filter $H_p(f)$. This leads us to the expression of the power:

$$P_0 = \int_{-1/2}^{+1/2} |H_p(f)|^2 S_X(f) df$$

A system without pre-emphasis/de-emphasis amounts to the addition of noise, with the power $\int_{-1/2}^{+1/2} S_B(f)df$, to the useful signal. For an accurate comparison of the systems with and without pre-emphasis, we have to consider that the power available to the system without pre-emphasis is equal to P_0. Therefore, the signal-to-noise ratio has the expression:

$$\rho = \frac{P_0}{\int_{-1/2}^{+1/2} S_B(f)df} = \frac{\int_{-1/2}^{+1/2} |H_p(f)|^2 S_X(f)df}{\int_{-1/2}^{+1/2} S_B(f)df}$$

3.

$$g = \frac{\rho_{PD}}{\rho} = \frac{\int_{-1/2}^{+1/2} S_B(f)df}{\int_{-1/2}^{+1/2} |H_p(f)|^2 S_X(f)df} \times \frac{\int_{-1/2}^{+1/2} S_X(f)df}{\int_{-1/2}^{+1/2} |H_d(f)|^2 S_B(f)df}$$

4. Using the Schwarz inequality, we get a lower-bound for the denominator, then the fact that $H_p(f)H_d(f) = 1$ leads us to:

$$g \leq \frac{\int_{-1/2}^{+1/2} S_B(f)df \int_{-1/2}^{+1/2} S_X(f)df}{\left|\int_{-1/2}^{+1/2} \sqrt{S_X(f)S_B(f)}df\right|^2} \tag{13.4}$$

The two sides are equal if and only if $|H_p(f)|^2 S_X(f)$ is proportional to $|H_d(f)|^2 S_B(f)$. Finally, we infer that the maximum of the gain g is obtained for:

$$|H_p(f)|^2 = \sqrt{\frac{S_B(f)}{S_X(f)}}$$

The corresponding gain is given by the right-hand side of equation 13.4 which is always greater than 1 according to the Schwarz inequality. As a conclusion, we started out with the expressions of the spectra and ended up with improved performances.

In many cases, the spectra of the signal and of the noise are such that a low-pass filter provides a good approximation of the de-emphasis filter. Therefore, a high-pass filter is used on the pre-emphasis side. To get a better idea, imagine that the treble level is “raised”, hence the name *pre-emphasis*.

Frequency modulation radio transmissions use this technique. It is also found in *speech coders*, such as the one used in exercise 12.4, page 469. In that case, the pre-emphasis filter used is very simple: it is a high-pass FIR filter with the two coefficient impulse response $h = [1 \quad -0.9375]$ (see the `code.m` program on page 683).

H8.5 (Estimation of an FIR filter's impulse response) (see page 301)

1. We saw in expression 8.45 that $R_{YX}(k) = h(k) \star R_{XX}(k)$. Because $h(k)$ has a finite length L:

$$R_{YX}(k) = h(0)R_{XX}(k) + \cdots + h(L-1)R_{XX}(k-L+1)$$

2. If we stack the L equations for k from 0 to $(L-1)$, we get the matrix expression:

$$\mathbf{r}_{YX} = \mathbf{R}_{XX}\mathbf{h}$$

with:

$$\begin{cases} \mathbf{h} = [h(0)\ h(1)\ \cdots\ h(L-1)]^T \\ \mathbf{r}_{YX} = [\mathbb{E}\{Y(n)X(n)\}\ \cdots\ \mathbb{E}\{Y(n)X(n-L+1)\}]^T \\ \mathbf{R}_{XX} = [\mathbb{E}\{X(n-i)X(n-j)\}]_{1\leq i,j\leq L} = R_{XX}(j-i) \end{cases}$$

Therefore, $\mathbf{h} = \mathbf{R}_{XX}^{-1}\mathbf{r}_{YX}$. Because $\mathbf{R}_{XX}$ is a Toeplitz matrix, the Levinson algorithm (see exercise 8.6) provides a fast technique for computing the inverse of $\mathbf{R}_{XX}$. Here we will only be using the MATLAB® function `inv`.

3. To estimate $R_{XX}(k)$ and $R_{YX}(k)$ based on a sequence of N observations, we first have to center $X(n)$ and $Y(n)$:

$$X_c(n) = X(n) - \frac{1}{N}\sum_{j=1}^{N} X(j) \text{ and } Y_c(n) = Y(n) - \frac{1}{N}\sum_{j=1}^{N} Y(j)$$

then we use:

$$\hat{R}_{XX}(k) = \frac{1}{N}\sum_{j=1}^{N-k} X_c(j+k)X_c(j) \text{ and } \hat{R}_{YX}(k) = \frac{1}{N}\sum_{j=1}^{N-k} Y_c(j+k)X_c(j)$$

Type:

```
%===== CREPIMP.M
N=26; Tx=length(x);
for jj=1:N
    rx(jj)=[x zeros(1,jj-1)]*[zeros(jj-1,1) ; x'];
    ryx(jj)=[y zeros(1,jj-1)]*[zeros(jj-1,1) ; x'];
end
Rxx=toepl(rx); hest=inv(Rxx) * ryx';
plot((0:N-1),hest,'x',(0:N-1),h,'o')
```

H8.6 (The Levinson algorithm) (see page 314)

1. Save the function:

```
function [ai,s2]=levinson(x,K)
%%=======================================%
%% Prediction: Levinson algorithm       %
%% SYNOPSIS: [ai,s2]=LEVINSON(x,K)      %
%%     x = Input sequence               %
%%     K = Model order                  %
%%    s2 = Prediction error             %
%%    ai = Model coeffts (1 a_1 ... a_K) %
%%=======================================%
N=length(x); x=x(:); x=x-ones(1,N)*x/N; R=zeros(1,K+1);
%===== Estimation of the covariances
for kk=1:K+1
    R(kk)=x(1:N-kk+1)'*x(kk:N)/N;
end
varE=zeros(K+1,1); ki=zeros(K+1,1); ai=zeros(K+1);
varE(1)=R(1,1); ai(1,1)=1;
%===== Levinson algorithm
for kk=2:K+1
    ki(kk)=-R(kk:-1:2)*ai(1:kk-1,kk-1)/varE(kk-1);
    ai(1:kk,kk)=[ai(1:kk-1,kk-1);0]+...
             ki(kk)*[0;conj(ai(kk-1:-1:1,kk-1))];
    varE(kk)=varE(kk-1)*(1-ki(kk)*ki(kk)');
end
s2=varE(K+1);
return
```

Type:

```
%===== TESTLEVINSON.M
clear; K=7; N=2000; w=randn(N,1)+j*randn(N,1);
avrai=[1;0.48-0.45*i;0.89-0.22*i;0.48-0.4*i;-0.01-0.22*i];
P=length(avrai)-1;
x=filter(1,avrai,w);
[aestL sest2L]=levinson(x,K);
%===== Direct estimation
[aestD,sest2D]=xtoa(x,K);
[[avrai;zeros(K-P,1)] aestD aestL(:,K+1)]
```

In order to test the **levinson** function, we took a complex AR process generated by a complex coefficient polynomial $A(z)$ such that $A(z) \neq 0$ for $|z| \geq 1$. In the case of a complex process, the estimation of the covariances has the expression:

$$\widehat{R}(k) = \frac{1}{N}\sum_{n=1}^{N-k+1} X_c(n+k)X_c^*(n)$$

We used the **xtoa.m** function written on page 330.

2. The Levinson algorithm allows us to solve the equation $\mathbf{R}[1\ a_1 \dots a_p]^T = [\sigma^2\ 0 \cdots 0]^T$ where $\mathbf{R}$ is a $(p+1)\times(p+1)$ Toeplitz matrix. This algorithm also applies to solving the equation:

$$\mathbf{R}\mathbf{h} = \mathbf{c}$$

where $\mathbf{R}$ is once again a Toeplitz matrix, but where $\mathbf{c}$ is any vector. We will encoutner this type of equation in section 11.2.5 when we identify filters (see, for example, equations 11.22 and 11.42 in Chapter 11). $\mathbf{R}$ is an autocovariance matrix, $\mathbf{c}$ is a covariance vector between the input and the output, and $\mathbf{h}$ is the FIR filter we have to identify.

As we did previously, we are going to try to find an iterative algorithm that updates at step n the length $(n-1)$ solution $\mathbf{h}^{(n-1)}$ found at step $(n-1)$. This solution $\mathbf{h}^{(n-1)}$ is such that $\mathbf{R}^{(n-1)}\mathbf{h}^{(n-1)} = \mathbf{c}^{(n-1)}$, where $\mathbf{R}^{(n-1)}$ is the $(n-1)\times(n-1)$ Toeplitz matrix and $\mathbf{c}^{(n-1)} = (c(1),\dots,c(n-1))^T$. Because $\mathbf{R}^{(n)}$ is a Toeplitz matrix, we can write:

$$\mathbf{R}^{(n)}\begin{pmatrix}\mathbf{h}^{(n-1)}\\ 0\end{pmatrix} = \begin{pmatrix}\mathbf{R}^{(n-1)} & \mathbf{r}(n)\\ \mathbf{r}^T(n) & R(0)\end{pmatrix}\begin{pmatrix}\mathbf{h}^{(n-1)}\\ 0\end{pmatrix} = \begin{pmatrix}\mathbf{c}^{(n-1)}\\ \mathbf{r}^T(n)\mathbf{h}^{(n-1)}\end{pmatrix}$$

At step n, we need to find the solution $\mathbf{h}^{(n)}$ such that:

$$\begin{pmatrix}\mathbf{R}^{(n-1)} & \mathbf{r}(n)\\ \mathbf{r}^T(n) & R(0)\end{pmatrix}\mathbf{h}^{(n)} = \begin{pmatrix}\mathbf{c}^{(n-1)}\\ c(n)\end{pmatrix}$$

If we substract the two matrix equations, we get:

$$\begin{pmatrix} \mathbf{R}^{(n-1)} & \mathbf{r}(n) \\ \mathbf{r}^T(n) & R(0) \end{pmatrix} \Delta\mathbf{h}^{(n)} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{r}^T(n)\mathbf{h}^{(n-1)} - c(n) \end{pmatrix}$$

Notice that this equation is analogous to the one we solved with the Levinson recursion. The algorithm therefore consists of applying this recursion to the updating terms $\Delta\mathbf{h}^{(n)}$. This is what is done in the following function:

```
function h=invToeplitz(rxx,rxy)
%%==========================================%
%% SYNOPSIS: h=INVTOEPLITZ(rxx,rxy)         %
%%    rxx = (NxN) autocovariance matrix     %
%%    rxy = (Nx1) covariances between x and y %
%%    h   = (Nx1)vector                     %
%%==========================================%
%===== Initial values : Rxx(0) h(0,0) = rxy(0)
NbCor=length(rxy); h=zeros(NbCor,1); h(1)=rxy(1)/rxx(1);
delta(1)=0; delta(2)=1/rxx(1);
%=====
for L=1:NbCor-1
    p=0;
    for I=1:L, p=p+rxx(I+1)*delta(I+1); end
    beta=1/(p*p-1); alpha=beta*p;
    for I=0:L
        deltaP(I+1)=alpha*delta(L-I+1)-beta*delta(I+1);
    end
    for I=0:L, delta(I+2)=deltaP(I+1); end
    %=====
    q=0;
    for I=1:L, q=q-h(L-I+1)*rxx(I+1); end
    q=q+rxy(L+1);
    for I=0:L, h(I+1)=h(I+1)+q*delta(I+2); end
end
return
```

To test this function, the following program constructs a Toeplitz matrix **R** (denoted by `RXX` in the program) and a vector **c** (denoted by `rxy`) based on an impulse response denoted by **h**:

```
%===== TESTINVTOEPL.M
h=[1;0.3+j;-0.2-6*j;0.1;0.07;0.9]; lh=length(h);
%===== Positive Toeplitz matrix (lh x lh)
N=100; w=randn(N,1); x=filter(0.5*ones(2,1),1,w);
```

```
for ii=1:lh
    cxx(ii)=x(1:N-ii+1)'*x(ii:N);
end
RXX=toeplitz(cxx); rxy=RXX*h;
ha=invToepl(cxx,rxy); [h ha]
```

H9 Continuous spectra estimation

H9.1 (Spectrum estimation using the Welch method) (see page 327)

1. `welch` function:

```
function [sf,gamma]=welch(x,lgfen,typef,Lfft,beta)
%%=====================================================%
%% SYNOPSIS: [sf,gamma]=WELCH(x,lgfen,typef,Lfft,beta) %
%%     x     = Input sequence                          %
%%     lgfen = Analysis window                         %
%%     typef = Window: h(ham) or r(rect)               %
%%     Lfft  = FFT length                              %
%%     beta  = Confidence parameter                    %
%%     gamma = Confidence interval (100*beta%)         %
%%     sf    = Spectrum                                %
%%=====================================================%
x=x(:); N=length(x); sf=zeros(Lfft,1);
%===== Suppression of the trend
[abid x]=tendoff(x);
Ks2=fix(lgfen/2); lgfen=2*Ks2; nbblocks=fix(N/Ks2)-1;
if (typef(1)=='h')
    fen=0.54-0.46*cos(2*pi*(0:lgfen-1)/lgfen);
    elseif (typef(1)=='r')
        fen=ones(1,lgfen);
    else
        disp('Unknown window')
        return
end
wfen=(1/sqrt(fen*fen'))*fen';
for tt=1:nbblocks
    ti=(tt-1)*Ks2+1;tf=ti+lgfen-1; pt=x(ti:tf) .* wfen;
    apf=abs(fft(pt,Lfft)) .^2; sf=sf+apf;
end
sf=sf/nbblocks; gamma=sqrt(2)*erfinv(beta)/sqrt(nbblocks);
return
```

2. The `welch` function is tested and compared with the `smperio` function in the following program:

```
%===== CTESTWELCH.M
clear all
```

```
T=1000; blocana=64; Lfft=1024; freq=(0:Lfft-1)/Lfft;
%===== Signal generation
w=randn(1,T); x=filter([1 .5 .02 .01],1,w);
xfth=20*log10(abs(fft([1;.5;.02;.01],Lfft)));
[xfw gamma]=welch(x,blocana,'ham',Lfft,0.95);
[xfp freqp]=smperio(x,16,'t');
xfw=10*log10(xfw); xfp=10*log10(xfp);
df1=-10*log10(1+gamma); df2=-10*log10(1-gamma);
plot(freq,xfw,freq,xfw+df1,'g-.',freq,xfw+df2,'g-.',freq,xfth);
set(gca,'xlim',[0 0.5]);
hold on; plot(freqp,xfp,'r'); hold off
```

H9.2 (Estimating the spectrum of a binary signal) (see page 327)
Type:

```
%===== SPECBIN.M
clear; lb=1024; A=5; T1=10; T2=25; T=T1+T2;
gn=[A*ones(T1,1);zeros(T2,1)]; an=sign(rand(1,lb)-1/2);
yn=gn*an; yn=reshape(yn,1,T*lb);
%===== Signal
figure(1),plot(yn(1:7*T)); grid
set(gca,'ylim',1.2*[-A A],'xlim',[1 7*T])
%===== Spectra
Lfft=1024; fq=(0:Lfft-1)/Lfft; tbloc=128;
spectheor=abs(fft(gn,Lfft)) .^2 /T;
[syf gamma]=welch(yn,tbloc,'ham',Lfft,95);
figure(2); subplot(212); plot(fq,10*log10(syf)); grid
set(gca,'xlim',[0 1/2],'ylim',[-40 20])
subplot(211); plot(fq,10*log10(spectheor),'r'); grid
set(gca,'xlim',[0 1/2],'ylim',[-40 20])
%===== Theoretical spectrum
absGf=A*abs(sin(pi*(1:Lfft-1)*T1/Lfft) ./ sin(pi*(1:Lfft-1)/Lfft));
spectheor2=[A*T1 absGf] .^2/T;
```

Both the theoretical and the estimated spectra are represented in Figure H9.1. We can also use the `smperio` function (example 9.1), by typing:

```
[syfp, freqp]=smperio(yn);
figure(2); subplot(212); hold on;
plot(freqp,10*log10(syfp),'g'); hold off
```

With the values chosen for the size of the windows, the results are noticeably better with the `smperio` function. Note, however, that the Welch method calculates several (559) length 1,024 FFTs whereas the smoothed periodogram method calculates only one FFT with a length at least equal to 35,840 in our example.

H9.3 (Spectral observation and oversampling) (see page 328)
Type (Figure H9.2):

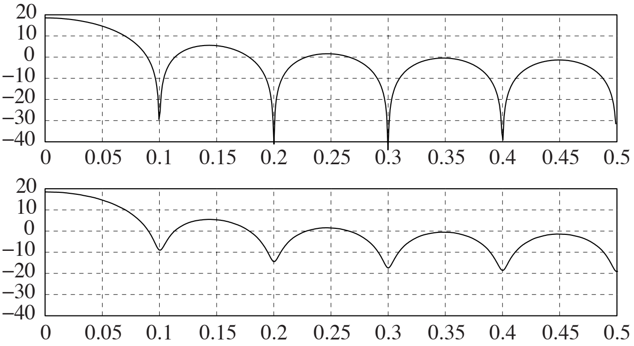


Figure H9.1 – *Signal spectra: above, the theoretical spectrum, below, the estimated spectrum*

```
%===== specoversmp.m
M=8; Npts=1024; Lfft=256; freq=(0:Lfft-1)/Lfft;
%===== Signal generation
NB=8; hn=rif(NB,1/4); xsig=filter(hn,1,randn(1,Npts));
xsigs=welch(xsig,16,'ham',Lfft,0.95);
subplot(311); plot(freq,xsigs); axis([0 .5 0 max(xsigs)]);
%===== Insertion of zeros
y=zeros(M,Npts); y(1,:)=xsig; yr=zeros(1,Npts*M); yr(:) = y;
yrs=welch(yr,128,'ham',Lfft,0.95);
subplot(312); plot(freq,yrs); axis([0 .5 0 max(yrs)]);
%===== Lowpass filtering
NPB=50; unsur2M=1/(M*2); hinter=rif(NPB,unsur2M);
y=filter(hinter,1,yr);
ys=welch(y,128,'ham',Lfft,0.95);
subplot(313); plot(freq,ys); axis([0 .5 0 max(ys)]);
```

Notice that the window size used for the estimation of the spectrum is 8 times smaller in the first case than it is in the next two, in order to have the same resolution for the three spectra.

H9.4 (The Burg estimation of the AR parameters) (see page 335)

1. Save the function:

```
function [ai,s2]=burg(x,K)
%%=======================================%
%% SYNOPSIS: [ai, s2]=BURG(x,K)          %
%%    x  = Input sequence                %
%%    K  = Prediction order              %
%%    s2 = Prediction error              %
%%    ai = Model coeffts (1 a_1  ...  a_K) %
```

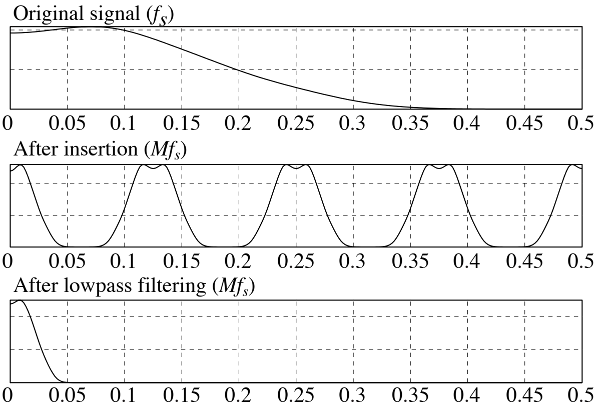


Figure H9.2 – *Spectra of the generated signals*

```
%%=========================================%
Nx=length(x); x=x(:)'; % Row vector
x=x-x*ones(Nx,1)/Nx;
varE=zeros(K+1,1);
kburg=zeros(K+1,1); ai=zeros(K+1,K+1);
epsf=x(Nx:-1:1); epsb=conj(x(Nx:-1:1));
varE(1)=x*x'/Nx; ai(1,1)=1;
for mm=2:K+1
    le =length(epsf);
    num=-2*epsb(1:le-1)*conj(epsf(2:le)');
    den=epsb(1:le-1)*epsb(1:le-1)'+epsf(2:le)*epsf(2:le)';
    kburg(mm)=num/den; ai(1,mm)=1;
    for nn=2:mm
        ai(nn,mm)=ai(nn,mm-1)+kburg(mm)*ai(mm-nn+1,mm-1)';
    end
    varE(mm)=varE(mm-1)*(1-kburg(mm)*kburg(mm)');
    auxf=epsf(2:le) + kburg(mm)*conj(epsb(1:le-1));
    epsb=epsb(1:le-1) + kburg(mm)*conj(epsf(2:le));
    epsf=auxf;
end
ai=ai(:,K+1); s2=varE(K+1);
return
```

2. The program `testburg.m` can be used to compare the results obtained by direct resolution or by using the Burg algorithm:

```
%===== TESTBURG.M
%===== P-order AR
clear; N=20000; w=randn(N,1)+3*j*randn(N,1);
avrai=[1;0.48-0.45*i;0.89-0.22*i;0.48-0.4*i;-0.01-0.22*i];
P=length(avrai)-1;
```

```
x=filter(1,avrai,w);
K=7; % Over-estimated order if K>P
%===== Estimation using Burg
[aestB,sest2B]=burg(x,K);
%===== Estimation using Levinson
[aestL,sest2L]=levinson(x,K);
%===== Direct estimation
[aestD,sest2D]=xtoa(x,K);
[[avrai;zeros(K-P,1)] aestD aestL(:,K+1) aestB]
```

H9.5 (AR-1 estimation and confidence intervals) (see page 336)

1. As we saw in exercise 7.1, the $100\alpha\%$ confidence ellipse for a two dimension Gaussian distribution with the mean $\mathbf{m}$ and the covariance matrix $\mathbf{C}$ is given by equation $(\mathbf{X}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{X}-\mathbf{m}) = -2\log(1-\alpha)$.

2. Type (Figure H9.3):

```
%===== CAR1STAT.M
clear; s2=1; a=-0.7; N=1000; nbe=500; alp=98/100;
calp=-2*log(1-alp); C=[N/(2*s2*s2) 0; 0 N/(1-a*a)];
ellipse([s2 a],C,calp); hold on
plot([s2 s2],a*[.8 1.2],'-'); plot(s2*[.8 1.2],[a a],'-')
for ii=1:nbe
    wn=s2*randn(1,N); xn=filter(1,[1 a],wn);
    r0est=xn*xn'/N; r1est=xn(2:N)*xn(1:N-1)'/(N-1);
    aest=-r1est/r0est; s2est=r0est+aest*r1est;
    plot(s2est,aest,'+');
end
axis([ s2*.8 s2*1.2 a*1.2 a*0.8]); grid; hold off
```

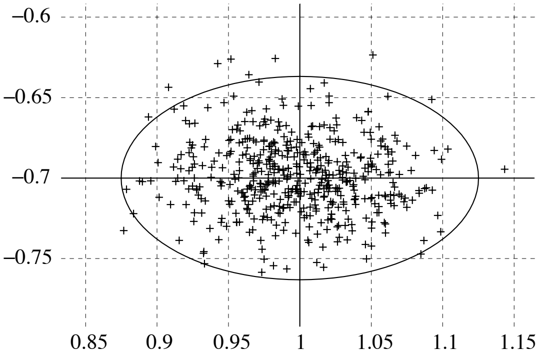


Figure H9.3 – 98% *confidence ellipse*

H10 Discrete spectra estimation

H10.1 (The Prony method) (see page 363)

1. According to property 10.1, the real signal $s(n)$ satisfies a recursive equation:

   $$s(n) + b_1 s(n-1) + \cdots + b_{2P} s(n-2P) = 0$$

   such that the $2P$ roots of the real coefficient, $2P$ degree polynomial $B(z) = z^{2P} + b_1 z^{2P-1} + \cdots + b_{2P}$ come in pairs of complex conjugates on the unit circle. These roots are written $z_k = \exp(\pm 2j\pi f_k)$. This expression makes it possible to calculate the frequencies f_k based on the calculation of the roots of the polynomial $B(z)$.

2. If we start out with $x(n) + b_1 x(n-1) + \cdots + b_{2P} x(n-2P) = \varepsilon(n)$ and stack the $N - 2P$ equations obtained from $n = 2P$ to $n = N-1$, we get:

   $$\begin{aligned}
   &x(2P) + b_1 x(2P-1) + \cdots + b_{2P} x(0) = \varepsilon(2P)\\
   &x(2P+1) + b_1 x(2P) + \cdots + b_{2P} x(1) = \varepsilon(2P+1)\\
   &\vdots\\
   &x(N-1) + b_1 x(N-2) + \cdots + b_{2P} x(N-2P-1) = \varepsilon(N-1)
   \end{aligned}$$

   which can be written, with obvious notations $\mathbf{Db} = \mathbf{e}$. $\mathbf{D}$ is an $(N-2P) \times (2P+1)$ matrix constructed from the data and $\mathbf{b} = [1\ b_1\ \ldots\ b_{2P}]^T$ is the vector associated with the coefficients of the polynomial.

   In order to estimate $\mathbf{b}$, the Prony method suggests minimizing the scalar $\mathbf{e}^T\mathbf{e} = \mathbf{b}^T\mathbf{D}^T\mathbf{D}\mathbf{b}$ under the constraint stating that the first component of $\mathbf{b}$ must be equal to 1. If we choose $\mathbf{u} = [1 \quad 0 \quad \ldots \quad 0]^T$ and $\mathbf{R} = \mathbf{D}^T\mathbf{D}$ (a $(2P+1) \times (2P+1)$ square matrix), we have to solve:

   $$\left\{\begin{array}{l}\min(\mathbf{b}^T\mathbf{R}\mathbf{b})\\ \text{such that : } \mathbf{b}^T\mathbf{u} - 1 = 0\end{array}\right. \tag{13.5}$$

   Note that the matrix $\mathbf{R} = \mathbf{D}^T\mathbf{D}$ involved in this expression is the matrix we associated with the covariance method on page 296.

   The Lagrange multiplier method consists of substituting the problem formulated by expression 13.5 with the following equivalent problem:

   $$\left\{\begin{array}{l}\min\left\{\mathbf{b}^T\mathbf{R}\mathbf{b} - \lambda(\mathbf{b}^T\mathbf{u} - 1)\right\}\\ \text{such that : } \mathbf{b}^T\mathbf{u} - 1 = 0\end{array}\right.$$

If we differentiate the first expression with respect to $\mathbf{b}$, we get the equation $\mathbf{Rb} = \lambda\mathbf{u}$, which gives us $\mathbf{b} = \lambda\mathbf{R}^{-1}\mathbf{u}$. If we take into account the constraint, we have $\lambda\mathbf{u}^T\mathbf{R}^{-1}\mathbf{u} - 1 = 0$, based on which we can determine λ. In the end, we get:

$$\mathbf{b} = \frac{1}{\mathbf{u}^T\mathbf{R}^{-1}\mathbf{u}}\mathbf{R}^{-1}\mathbf{u} \tag{13.6}$$

Here is a summary of the frequency estimation algorithm:

Steps:

- Construct the matrix $\mathbf{D}$ then calculate $\mathbf{R} = \mathbf{D}^T\mathbf{D}$.
- Calculate $\mathbf{b} = \dfrac{1}{\mathbf{u}^T\mathbf{R}^{-1}\mathbf{u}}\mathbf{R}^{-1}\mathbf{u}$.
- Theoretically, the polynomial constructed from the elements of $\mathbf{b}$ has all its roots on the unit circle. They can be obtained either by direct computation with the `roots` function, or by an FFT evaluation of the expression:

$$G(f) = \frac{1}{|1 + b_1 e^{-2j\pi f} + \cdots + b_{2P} e^{-4Pj\pi f}|}$$

and determine its maxima, which theoretically, should be infinite.

3. Type the program:

```
%===== CPRONY.M
clear
%===== Original signal
Am=[2 1.5 1]; P=length(Am); F=[0.2 0.225 0.3];
N=25; s=Am*cos(2*pi*F'*(0:N-1));
SNR=40; sigma2= (s*s'/N)/(10 ^(SNR/10));
%===== Noisy signal
x=s+sqrt(sigma2)*randn(1,N);
D=toepl(x(2*P+1:N),x(2*P+1:-1:1));
R=D*D'; U=[1;zeros(2*P,1)];
B=inv(R)*U;
lambda=U'*B; A=B/lambda;
%===== Verification
Lfft=256; fq=(0:Lfft-1)/Lfft;
gf=1 ./ abs(fft(A,Lfft));
subplot(211); plot(fq(1:Lfft/2),gf(1:Lfft/2)); grid
xf=abs(fft(x,Lfft));
subplot(212); plot(fq(1:Lfft/2),xf(1:Lfft/2)); grid
```

The result shown in Figures H10.1 indicates that the frequencies 0.2 and 0.225, having a difference of 0.025 (smaller than the Fourier limit which has the same order of magnitude as $1/N = 0.04$) can still be distinguished on the first spectrogram, corresponding to the Prony method, while they cannot on the second one obtained by a direct computation of the periodogram. This only illustrates, rather than prove, the superiority of one method over the other. The only way of proving it would be to compare performances in the presence of noise, by conducting for example a simulation based on a noisy signal with known frequencies, and by calculating the square deviations around the true values.

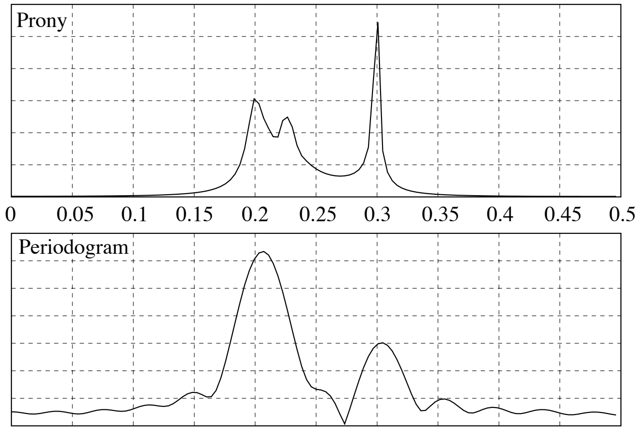


Figure H10.1 – *Above: spectrum obtained with the Prony method; below: periodogram. The signal-to-noise ratio is equal to* 40 *dB. The sample size is* $N = 25$*. The frequencies are* [0.2 0.225 0.3] *and the amplitudes are* [2 1.5 1] *respectively*

If the signal-to-noise ratio decreases, even by as little as 10 dB, the Prony method leads to significantly worse results.

H10.2 (The Pisarenko method) (see page 364)

1. If we expand $\mathbb{E}\{x(n+k)x(n)\}$, and use the fact that there is no correlation between $s(n;a)$ and $b(n)$, and that $b(n)$ is centered, we get, for the autocovariance function of $x(n)$:

$$R_{xx}(k) = \mathbb{E}\{s(n+k)s(n)\} + \mathbb{E}\{b(n+k)b(n)\} = R_{ss}(k) + R_{bb}(k)$$

Therefore, $R_{xx}(k) = R_{ss}(k) + \sigma^2\delta(k)$ and:

$$\mathbf{R}_x = \mathbf{R}_s + \sigma^2\mathbf{I}$$

2. The rank of the matrix $\mathbf{R}_s$ is equal to $(M-G)$. This is equivalent to saying that the dimension of the kernel is equal to G and that the dimension of the complementary subspace is equal to $(M-G)$. Let $\{\mathbf{w}_1, \dots, \mathbf{w}_G\}$ be an orthonormal basis of the kernel and $\{\mathbf{v}_1, \dots, \mathbf{v}_{M-G}\}$ a basis of the orthogonal subspace. We then have $\mathbf{R}_s\mathbf{w}_j = \mathbf{0}$ and $\mathbf{R}_s\mathbf{v}_i = \lambda_i\mathbf{v}_i$.

   If we multiply the two sides of the equation $\mathbf{R}_x = \mathbf{R}_s + \sigma^2\mathbf{I}$ by a vector $\mathbf{w}_j$, we get $\mathbf{R}_s\mathbf{w}_j = \sigma^2\mathbf{w}_j$. Any vector of the noise subspace is therefore an eigenvector of $\mathbf{R}_x$ associated with the same eigenvalue σ^2.

   If we mutiply the two sides of the equation $\mathbf{R}_x = \mathbf{R}_s + \sigma^2\mathbf{I}$ by a vector $\mathbf{v}_i$, we get $\mathbf{R}_s\mathbf{v}_i = (\lambda_i + \sigma^2)\mathbf{v}_i$. Hence any vector of the signal subspace is an eigenvector of $\mathbf{R}_x$ associated with the eigenvalue $(\lambda_i + \sigma^2) > \sigma^2$.

   To sum it all up:

$$\begin{cases} \mathbf{R}_x\mathbf{w}_j = \sigma^2\mathbf{w}_j \\ \mathbf{R}_x\mathbf{v}_i = (\lambda_i + \sigma^2)\mathbf{v}_i \end{cases}$$

   The observation matrix $\mathbf{R}_x$ therefore has M positive eigenvalues arranged in the following order:

$$(\lambda_1 + \sigma^2) \geq \cdots \geq (\lambda_{M-G} + \sigma^2) > \sigma^2 = \cdots = \sigma^2$$

   The multiplicity of the smallest eigenvalue is equal to G.

3. Study of the stationarity:

$$\mathbb{E}\{s(n)\} = \mathbb{E}\{A\cos(2\pi f_0 n + \Phi)\} = \int_0^{2\pi} A\cos(2\pi f_0 n + \phi)\frac{d\phi}{2\pi} = 0$$

   Likewise, we have:

$$\begin{aligned} R_{ss}(k) &= \mathbb{E}\{s(n+k)s(n)\} \\ &= \mathbb{E}\{A\cos(2\pi f_0(n+k) + \Phi)A\cos(2\pi f_0 n + \Phi)\} \\ &= \frac{A^2}{2}\mathbb{E}\{\cos(2\pi f_0 k) + \cos(2\pi f_0(2n+k) + 2\Phi)\} \\ &= \frac{A^2}{2}\cos(2\pi f_0 k) + \frac{A^2}{2}\int_0^{2\pi}\cos(2\pi f_0(2n+k) + 2u)\frac{du}{2\pi} \\ &= \frac{A^2}{2}\cos(2\pi f_0 k) \end{aligned}$$

   Hence $s(n)$ is second order wide sense stationary process.

We saw on page 359 that a mixture of P real sines satisfies a $2P$ order recursive equation. Here we can directly check that $R_{ss}(k) = R_{ss}(0)\cos(2\pi f_0 k)$ satisfies, for any k, the recursive equation:

$$R_{ss}(k) - 2z_0 R_{ss}(k-1) + R_{ss}(k-2) = 0 \text{ where } z_0 = \cos(2\pi f_0)$$

If we take $k = 0$, 1 and 2, and stack everything, we get the matrix:

$$\mathbf{R}_s = \begin{bmatrix} R_{ss}(0) & R_{ss}(-1) & R_{ss}(-2) \\ R_{ss}(1) & R_{ss}(0) & R_{ss}(-1) \\ R_{ss}(2) & R_{ss}(1) & R_{ss}(0) \end{bmatrix}$$

such that $\mathbf{R}_s\mathbf{a} = \mathbf{0}$ if $\mathbf{a} = [1 \quad -2z_0 \quad 1]^T$. The vector $\mathbf{a}$ is therefore a vector of the kernel $\mathbf{R}_s$, and because of what was said previously, the vector $\mathbf{a}$ is also an eigenvector of the matrix $\mathbf{R}_x$ associated with its *smallest eigenvalue.*

4. This result can easily be generalized to the case where $s(n)$ is the mixture of P real sines. The eigenvector associated with the smallest eigenvalue of the $(2P+1)$ order covariance matrix of the observation has, as its components, the coefficients of the recursive equation associated with the frequencies of the signal $s(n)$. This leads us to what is called the Pisarenko algorithm, used for measuring the P frequencies of a real signal:

Steps:

- *the* $(2P+1)$ *order covariance matrix* $\mathbf{R}_x$ *is estimated based on the data;*
- *the eigendecomposition of* $\mathbf{R}_x$ *is performed. It gives us an eigenvector* $\mathbf{v}$ *associated with the smallest eigenvalue;*
- *the* $2P$ *degree polynomial* $A(z)$ *is constructed based on* $\mathbf{v}$. *The* $2P$ *complex conjugate roots* z_k *are extracted from it, which leads us to the frequencies* $f_k = \frac{1}{2\pi}\arg(z_k)$.

Rather than to extract the roots using the `roots` function, we can take advantage of the fact that these roots belong *a priori* to the unit circle. All we need to do is compute the polynomial $A(z)$ on the unit circle, which is done by computing the FFT applied to its coefficients on a large number of points, then to search for the minima, which are not exactly equal to zero, because of the noise.

5. The following program implements this algorithm:

```
%===== CPISAR.M
Lfft=256; freq=(0:Lfft-1)/Lfft;
N=25; Am=[2 1.5 1]; F=[0.2 0.225 0.3]; P=length(Am);
%===== Original signal
s=Am*cos(2*pi*F'*(0:N-1));
SNR=40; sigma2= (s*s'/N)/(10 ^(SNR/10));
%===== Noisy signal
x=s+sqrt(sigma2)*randn(1,N);
D=toepl(x(2*P+1:N),x(2*P+1:-1:1)); R=D*D';
[Vvect Val]=eig(R); [ordVal iV]=min(diag(Val)) ;
Vvect=Vvect(:,iV); vf=1 ./ abs(fft(Vvect(:,1),Lfft));
plot(freq(1:Lfft/2),vf(1:Lfft/2)); grid
```

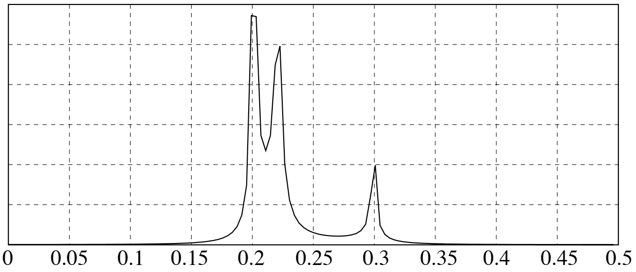


Figure H10.2 – *Pisarenko separation of the frequencies*

Figure H10.2 shows the result. Three maxima are clearly visible near the frequencies contained in the noiseless test signal. Using the Prony method in the same conditions, in exercise 10.1, we had maxima that were not as clear cut as the ones obtained here. However we should not conclude and say that the Prony methdo is less efficient than the Pisarenko method, because if we apply a bijective function such as $y = 1/x$ to the variable x, we can easily change a hardly visible maximum into a maximum high amplitude. Of course, in no way does this improve the performances in the presence of noise. The right way to compare these performances is to evaluate the square deviation between the estimated frequencies and the "true" frequencies as a function of the level of noise. The program `Ccomppp.m` can easily be used to to compare the two methods through simulation:

```
%===== CCOMPPP.M
N=25;Am=[2 1.5 1];F=[0.2 0.225 0.3];
P=length(Am); s=Am*cos(2*pi*F'*(0:N-1));
SNR=40;sigma2= (s*s'/N)/(10 ^(SNR/10));
```

```
x=s+sqrt(sigma2)*randn(1,N);
D=toepl(x(2*P+1:N),x(2*P+1:-1:1)); R=D*D';
%===== Prony
U=[1;zeros(2*P,1)];B=inv(R)*U;lambda=U'*B;
A=B/lambda; fkProny=angle(roots(A))/(2*pi);
FestProny=sort(fkProny(find(fkProny>0)))';
%===== Pisarenko
[Vvect Val]=eig(R); [ordVal iV]=min(diag(Val)) ;
Vmin=Vvect(:,iV); fkPisar=angle(roots(Vmin))/(2*pi);
FestPisar=sort(fkPisar(find(fkPisar>0)))';
F, FestProny, FestPisar
```

As we have already observed, the frequencies can be estimated either by using the `roots` function or the corresponding polynomial in the unit circle by way of the `fft` function, and determine the maxima of the FFT's modulus. In the program, we chose the first method as it avoids the search for the maxima.

H10.3 (MUSIC 2D) (see page 388)

1. The 2D-MUSIC function is given by

$$S_{\text{MUSIC}}(\zeta,\phi)=\frac{1}{\mathbf{a}^H(\zeta,\phi)\mathbf{G}\mathbf{G}^H\mathbf{a}(\zeta,\phi)}$$

   where $\mathbf{G}\mathbf{G}^H$ refers to the orthogonal projector onto the noise subspace.

2. Type:

```
%===== MUSIC2D.m
clear all
N=100; K=3; M=25; sigma=0.3;
%===== dzeta from 0 to 180, phi from -180 to 180
dzeta=[30 40 70]; phi=[60 50 20];
%===== Array sensor location
loc_array=zeros(3,M);
for xx=1:5
    for yy=1:5
        loc_array(1:2,(xx-1)*5+yy)=[xx-1,yy-1]'/2-1;
    end
end
Aalpha=array2D(dzeta,phi,loc_array); sn=randn(K,N);
bn=sigma*(randn(M,N)+j*randn(M,N)); xn=Aalpha*sn+bn;
%===== Covariance estimation
Rest=zeros(M);
for nn=1:N, Rest=Rest + xn(:,nn)*xn(:,nn)'; end
Rest=Rest/N; [Uest,Dest,Vest]=svd(Rest);
%===== Noise subspace projector
```

```
Vnoise=Vest(:,(K+1):M); PIest=Vnoise*Vnoise';
plage_theta=(20:0.5:90);ltheta=length(plage_theta);
plage_phi=(10:0.5:80);lphi=length(plage_phi);
imageM=zeros(ltheta,lphi);
for iphi=1:lphi
    ph=plage_phi(iphi);
    for itheta=1:ltheta
        th=plage_theta(itheta);
        aij = array2D(th,ph,loc_array);
        imageM(itheta,iphi)=-log10(abs(aij'*PIest*aij));
    end
end
imagesc(plage_phi,plage_theta,imageM); grid
xlabel(['\phi = [' sprintf('%3.2g',phi) ']']);
ylabel(['\dzeta = [' sprintf('%3.2g',dzeta) ']']);
```

```
function A=array2D(dzeta,phi,loc_array)
%%=========================================%
%% SYNOPSIS: A=ARRAY2D(dzeta,phi,loc_array) %
%%=========================================%
DtoRAD=pi/180; M=size(loc_array,2); K=length(dzeta);
A=zeros(M,K);
for kk=1:K
    tk=dzeta(kk)*DtoRAD; pk=phi(kk)*DtoRAD;
    beta_kk=[cos(pk)*sin(tk);sin(pk)*sin(tk);cos(tk)];
    A(:,kk)=exp(2*pi*j*(loc_array.')*beta_kk);
end
return
```

H11 The least squares method

H11.1 (Determining the FT using the gain) (see page 400)

1. We can write $B(e^{2j\pi f}) = H(e^{2j\pi f})A(e^{2j\pi f})$ where $A(z) = 1 + a_1 z^{-1} + \cdots + a_p z^{-p}$ and $B(z) = b_0 + b_1 z^{-1} + \cdots + b_q z^{-q}$. If we use this expression for $f \in \{f_1, \ldots, f_N\}$, group everything together in matrix form, and define $z_k = e^{2j\pi f_k}$, we get:

$$\begin{bmatrix} 1 & z_1^{-1} & \cdots & z_1^{-q} & -H(z_1)z_1^{-1} & \cdots & -H(z_1)z_1^{-p} \\ \vdots & & & & & & \\ 1 & z_k^{-1} & \cdots & z_k^{-q} & -H(z_k)z_k^{-1} & \cdots & -H(z_k)z_k^{-p} \\ \vdots & & & & & & \\ 1 & z_N^{-1} & \cdots & z_N^{-q} & -H(z_N)z_N^{-1} & \cdots & -H(z_N)z_N^{-p} \end{bmatrix} \begin{bmatrix} b_0 \\ \vdots \\ b_q \\ a_1 \\ \vdots \\ a_p \end{bmatrix} = \mathbf{Gc}$$

We now need to determine the vector $\mathbf{c}$ that minimizes the deviation between $\mathbf{Gc}$ which depends linearly on the quantities we have to estimate, and the vector we are trying to determine, whose components are the complex gains at the frequencies f_k:

$$\mathbf{h} = \begin{bmatrix} H(z_1) & \dots & H(z_k) & \dots & H(z_N) \end{bmatrix}^T$$

The solution, in the least squares sense, is given by:

$$\mathbf{c} = (\mathbf{G}^H\mathbf{G})^{-1}\mathbf{G}^T\mathbf{h}$$

2. In practice, we would like to constrain the previous solution so that the resulting filter is causal, stable, and minimum phase. To do so, we have to ensure that the poles and zeros are inside the unit circle. Because we do not have the solution to such a problem, we suggest making the previously obtained filter causal, stable, and minimum phase without changing the modulus of its complex gain. Remember that the modulus of the complex gain remains the same if we use an inversion operation to make all the poles and zeros go from outside the unit circle to inside it. This process is implemented in the function `htoz` which uses the `poly` and `roots` functions.

3. Type:

```
function [b,a]=hftoz(H,F,qN,pD,r,W)
%%==========================================================%
%% Identification of the transfer function                  %
%% SYNOPSIS: [b,a]=HFTOZ(H,F,qN,pD,r,W)                     %
%%    (H,F) sequence of (complex gain, frequency)           %
%%    qN  = numerator's degree                              %
%%    pD  = denominator's degree                            %
%%    if r='real', the filter is real.                      %
%%    W   = weighting matrix diagonal                       %
%%    b   = coefficients of the numerator b=[b0 b1 .. b_q]  %
%%    a   = coefficients of the denominator a=[1 a1 .. a_p] %
%%==========================================================%
H=H(:); F=F(:); LH=length(H); LF=length(F);
if (LH~=LF) error('H et F do not have the same length'); end;
if (LF<(qN+pD+1)) error('Too high order'); end;
if (r=='real')                % H is symmetrized
    H=[H; conj(H(LF:-1:2))]; F=[F; 1-F(LF:-1:2)];
end;
j=sqrt(-1); mexp=exp(-2*j*pi*F*(0:max([qN pD])));
Gb=mexp(:,1:qN+1); Ga=-diag(H)*mexp(:,2:pD+1); G=[Gb Ga];
if (nargin==6)
    if (r=='real')
```

```
        G= diag(sqrt([W W(LF:-1:2)]))*G;
    else
        G=diag(sqrt(W))*G;
    end;
end;
C2=inv(G'*G)*G'*H; b=C2(1:qN+1);
a=[1; C2(qN+2:qN+pD+1)];
%===== Stable and minimum phase
ra=roots(a); va=find(abs(ra)>1);
ra(va)=1 ./ conj(ra(va)); a=poly(ra);
rb=roots(b); vb=find(abs(rb)>1);
rb(vb)=1 ./ conj(rb(vb)); b=poly(rb);
if (r=='real') a=real(a); b=real(b); end;
return
```

Note that if we choose a real filter, the program pads the sequence with values of the complex gain by hermitian symmetry. In that case, the values $\{f_1, \ldots, f_N\}$ have to belong to $(0, 1/2)$.

We added as an argument the weighting matrix $\mathbf{W}$, which is supposed to focus the effects of the least squares minimization on the passband and the stopband of the filter. This matrix is difficult to choose. In practice, the simplest matrix is diagonal, with 1 for the values in the passband and in the stopband, and 0 for those in the transition band. The resulting solution should theoretically be better. Another weighting would consist of using a matrix of the same type, but with $1/\delta_p$ for the passband and $1/\delta_a$ for the stopband.

4. The following program allows us to test the **hftoz.m** function for the design of a low-pass filter and of a derivative filter. Type:

```
%===== CTESTHFTOZ.M
Lfft=1024; freq=(0:Lfft-1)/Lfft;
num=10; den=2; F=(0:.01:0.5)'; LF=length(F);
%===== Passband
fcP=0.2; LfcP=ceil(2*fcP*LF);
%===== Ideal lowpass
H=[ones(LfcP,1);zeros(LF-LfcP,1)];
%===== Derivative
% H=j*pi*F;
%===== Weighting
% deltaP=.03; deltaA=.1;
%===== Weighting coeffts
% usP=1/deltaP; usA=1/deltaA;
%===== Stopband
% fcA=0.21; LfcA=ceil(2*fcA*LF);
% W=[usP*ones(1,LfcP) zeros(1,LfcA-LfcP) usA*ones(1,LF-LfcA)];
%=====
[b1 a1]= hftoz(H,F,num,den,'reel');
hf1=abs(fft(b1,Lfft)) ./ abs(fft(a1,Lfft));
hf1=20*log10(hf1/hf1(1));
plot(freq,hf1,'-',F,20*log10(H)); grid;
axis([0 .5 -40 10]);
```

Figure H11.1 shows the result for a low-pass filter with $p = 2$ and $q = 10$.

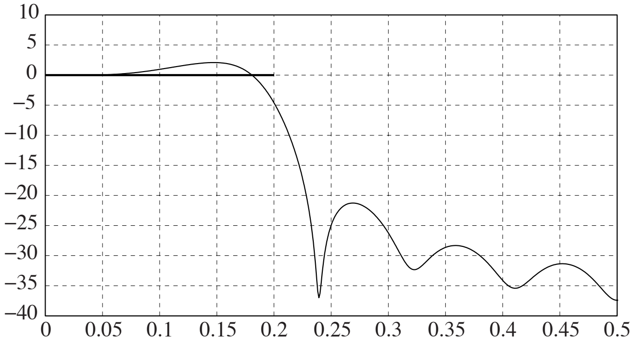


Figure H11.1 – *Designing an ideal low-pass filter*

Notice that, when the weighting is related to the ripples (comments in the program), the results are a little disappointing.

H11.2 (Approximating the inverse of a FIR filter) (see page 401)

1. The transfer function:

$$H(z) = 1 - 3.4z^{-1} + 1.2 = (1 - 3z^{-1})(1 - 0.4z^{-1})$$

has two zeros in $z_1 = 0.4$ et $z_2 = 3$. Its impulse response is of course $h(0) = 1$, $h(1) = -3$, $h(2) = 1.2$. The expression of its inverse is:

$$G(z) = \frac{1}{(1 - 3z^{-1})(1 - 0.4z^{-1})} = \frac{z^2}{(z - 3)(z - 0.4)}$$

Stability is ensured by associating $G(z)$ with the convergence area containing the unit circle, that is $0.4 < |z| < 3$. As a consequence, the impulse response has a causal part and an anti-causal part. If we use the partial fraction decomposition of $G(z)$ and the expansion of $1/(1 - u)$ for

$|u| < 1$, we can calculate the impulse response $\{g(n)\}$ explicitly. If we define $\alpha = 1/2.6$, we get:

$$\begin{aligned} G(z) &= z^2\left(\frac{\alpha}{z-3} + \frac{-\alpha}{z-0.4}\right) = -\frac{\alpha}{3}z^2\frac{1}{1-z/3} - \alpha z\frac{1}{1-0.4z} \\ &= -\frac{1}{7.8}z^2\left(1+3^{-1}z+\cdots+3^{-n}z^n+\cdots\right) \\ &\quad -\frac{1}{2.6}z\left(1+0.4z^{-1}+\cdots+(0.4)^n z^{-n}+\cdots\right) \end{aligned}$$

hence, if we identify $g(n)$ with the coefficient of z^{-n}, we have:

$$g(n) = \begin{cases} -\dfrac{1}{2.6}(0.4)^{n+1} & \text{for} \quad n \geq -1 \\ -\dfrac{1}{7.8}(3)^{n-2} & \text{for} \quad n < -1 \end{cases}$$

2. If we perform the convolution of $g(n)$ with $h(n)$, we get the sequence:

$$d(n) = h(0)g(n) + h(1)g(n-1) + h(2)g(n-2)$$

If we write this expression for n from $-A$ to $C+2$, we get:

$$\begin{cases} d(-A) &= h(0)g(-A) \\ d(-A+1) &= h(1)g(-A) + h(0)g(-A+1) \\ d(-A+2) &= h(2)g(-A) + h(1)g(-A+1) + h(0)g(-A+2) \\ & \vdots \\ d(0) &= h(2)g(-2) + h(1)g(-1) + h(0)g(0) \\ & \vdots \\ d(C) &= h(2)g(C-2) + h(1)g(C-1) + h(0)g(C) \\ d(C+1) &= h(2)g(C-1) + h(1)g(C) \\ d(C+2) &= h(2)g(C) \end{cases}$$

These expressions can be written in matrix form as $\mathbf{d} = \mathbf{Hg}$ where $\mathbf{H}$ is an $(N+A+C) \times (A+C+1)$ Toeplitz matrix constructed from $h(0)$, $h(1)$ and $h(2)$, where $N = 3$ represents the length of $\mathbf{h}$. Note that the column vectors of $\mathbf{H}$ are independent. Therefore, $\mathbf{H}$ has a full rank (meaning that the rank is $A+C+1$). In order to approximate the inverse filter, the vector $\mathbf{Hg}$ has to be as close as possible to the vector $\mathbf{d} = (\underbrace{0,\cdots,0}_{A}, 1, \underbrace{0,\cdots,0}_{N+C-1})^T$.

One solution is given by the pseudo-inverse:

$$\mathbf{g} = \mathbf{H}^{\#}\mathbf{d}$$

3. Type:

```
%===== IDENTDET.M
%===== Filter to be inverted
h=[1 -3.4 1.2]; N=length(h);
%===== Causal and anticausal length
C=20; A=10;
%===== Impulse 0 ... 0 1 0 ... 0
delta=zeros(N+A+C,1); delta(A+1)=1;
Hb=toeplitz([h zeros(1, A+C)],[h(1) zeros(1,A+C)]);
g=Hb \ delta;
%===== Verification using a convolution
stem(conv(g,h))
%===== Verification using the theoretical values
gA=-(3 .^ (0:-1:-A+2))/7.8; gC=-(0.4 .^(0:C+1))/2.6;
gth=[gA(A-1:-1:1) gC]';
[gth g], max(abs(gth-g))
```

Note that the closer the zeros outside the unit circle are to the unit circle, the higher A has to be. This goes for C and the zeros inside the unit circle as well.

H11.3 (Lattice filtering) (see page 416)

1. Type the function:

```
function [epsF, epsB]=lattice_analysis(xn, ki)
%%===============================================%
%% SYNOPSIS: [epsF, epsB]=LATTICE_ANALYSIS(xn,ki) %
%%    xn   = Signal                               %
%%    ki   = Reflection coefficients (k1 ... kP)  %
%%    epsF = Forward Error                        %
%%    epsB = Backward Error                       %
%%===============================================%
N=length(xn); epsF=zeros(N,1); epsB=zeros(N,1);
P=length(ki); eB=zeros(P,1); eBm1=zeros(P,1);
for nn=1:N
    eF=xn(nn);
    for pp=1:P
        eFp=eF+ki(pp)*eBm1(pp);
        eB(pp)=ki(pp)*eF+eBm1(pp);
        eF=eFp;
    end
    eBm1=[xn(nn); eB(1:P-1)];
    epsF(nn)=eFp;
    epsB(nn)=eB(P);
end
return
```

2. Type the function:

```
function [xn, epsB]=lattice_synthesis(epsF, ki)
%%==================================================%
%% SYNOPSIS: [epsF, epsB]=LATTICE_SYNTHESIS(xn,ki) %
%%    epsF = Forward Error                          %
%%    ki   = Reflection coefficients (k1 ... kP)    %
%%    xn   = Reconstructed signal                   %
%%    epsB = Backward Error                         %
%%==================================================%
N=length(epsF); xn=zeros(N,1); epsB=zeros(N,1);
P=length(ki); eB=zeros(P,1); eBm1=zeros(P,1);
for nn=1:N
    eF=epsF(nn);
    for pp=P:-1:1
        eF=eF-ki(pp)*eBm1(pp);
        eB(pp)=eBm1(pp)+ki(pp)*eF;
    end
    xn(nn)=eF;
    eBm1=[eF; eB(1:P-1)];
    epsB(nn)=eB(P);
end
return
```

3. Run the following program:

```
%===== TESTLATTICE.M
clear all
ai=[1 -1.8 0.9]; N=1000;
wn=randn(N,1); xn=filter(1,ai,wn);
ki=atok(ai);
[eF,eB]=lattice_analysis(xn,ki);
[xn_s, eB_s]=lattice_synthesis(eF,ki);
[max(abs([wn-eF])) max(abs([xn-xn_s]))]
figure(1); plot(xn);
hold on; plot(xn_s,'r'); hold off
figure(2); plot(eF);
hold on; plot(wn,'r'); hold off
figure(3); plot(eB,'b');
hold on; plot(eB_s,'r'); hold off
```

H11.4 (The LMS algorithm: channel identification) (see page 438)
The following program allows us to test the *LMS* algorithm:

```
%===== IDENTLMS.M
%===== Channel identification
h=[1 0.6 0.3]'; P=length(h); % Theoretical channel
N=4000;                      % Number of steps
%===== Signal generation
```

```
x=randn(N,1); v=filter(h,1,x); Pv=v'*v/N;
SNR=20; b=sqrt(Pv*10^(-SNR/10))*randn(N,1);
y=v+b;                          % Noisy observation
%===== LMS algorithm
mu=0.002;
hest=zeros(P,1); en=zeros(N-P+1,1);
for n=P:N
    en0=y(n) - hest'*x(n:-1:n-P+1);
    hest=hest + mu*en0*x(n:-1:n-P+1);
    en(n-P+1)=en0;
end
%===== Smoothing of the error over 200 points
en2=en .^2; moy=200; hmoy=ones(1,moy)/moy;
en2moy=filter(hmoy,1,en2(1:N-P+1));
endb=10 .* log10(en2moy(moy:N-P+1));
plot(endb); grid
[h hest]
```

The results are shown in Figure H11.2. As you can see, the final difference when the algorithm converges is directly related to the value of the signal-to-noise ratio.

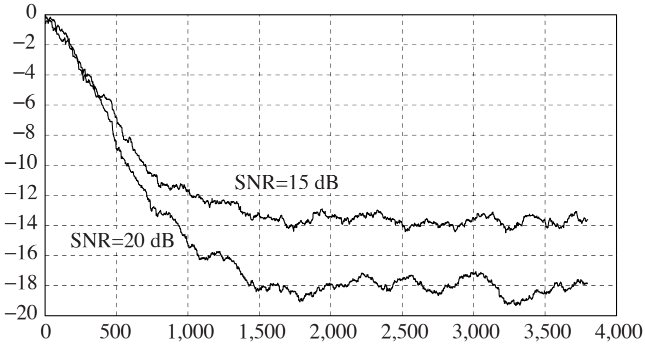


Figure H11.2 – *LMS algorithm: the error signal plotted against the number of iterations of the algorithm (square deviation in dB smoothed over* 200 *points)*

As we saw for the deterministic gradient, the higher the gradient step μ is, the higher the algorithm's descent speed. This is shown in Figure H11.3.

When μ increases, the error decreases faster to the final plateau. We can also show that the higher μ is, the higher the final plateau for the difference. This is called *misadjustment*. This behavior is the same for any adaptive algorithm: convergence speed comes at the cost of a higher misadjustment. However, this effect is rather faint in this case.

When comparing algorithms through simulation, the parameters are set so as to reach the same misadjustment level (convergence plateau) for both, then

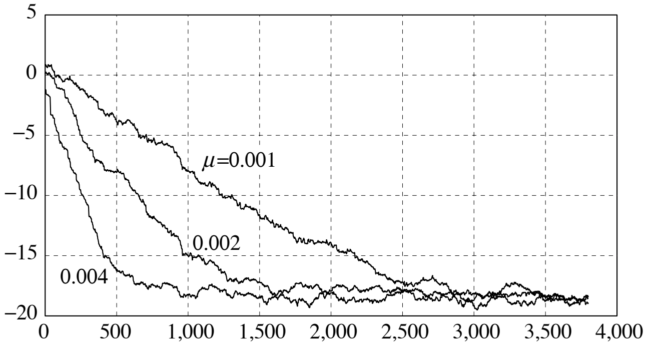


Figure H11.3 – *LMS algorithm: error signal plotted against the number of iterations of the algorithm for different values of μ*

the slopes of the error plots, which are characteristic of the convergence speed, are compared.

The theoretical maximum value of the deterministic gradient step is $\mu_{\max} = 2/\max_i(\lambda_i)$. In our example, $x(n)$ is a white noise with the variance 1, and therefore the matrix $\mathbf{R} = \mathbf{I}_2$. We then have $\mu_{\max} = 2$. As the simulations show, this value is much too high to ensure the convergence of the LMS algorithm. In practice, this is always the case, and μ is found experimentally, based on the observed data, by determining the value of μ that ensures convergence and then by reducing this value by 10%.

The LMS is adaptive: this means that it can adapt to the possible fluctuations the filter's coefficients could be subjected to. This goes beyond the strict context of linear filters, since the time invariance property is no longer obeyed. The tracking capability of the LMS algorithm can be measured by adding a break in the model and observing how the algorithm is able to track this break (see paragraph 11.5.2). Performances crucially depend on the choice of μ.

H12 Selected topics

H12.1 (Full-wave rectifier and simulation) (see page 451)

1. The full-wave rectifier is periodic with period $T_d = T_0/2$ where $T_0 = 1/F_0$. It is expandable in a Fourier series. The Fourier coefficients are:

$$\begin{aligned}X_n &= \frac{A}{T_d}\int_0^{T_d} \sin(2\pi F_0 t) e^{-2j\pi n t/T_d} dt\\ &= \frac{1}{T_d}\int_{-\infty}^{+\infty} x_T(t) e^{-2j\pi n t/T_d} dt = \frac{1}{T_d} X_T(n/T_d)\end{aligned}$$

where $x_T(t) = A\text{rect}_{T_d}(t - T_d/2)\sin(2\pi F_0 t)$. The Fourier transform of $x_T(t)$ is denoted by $X_T(F)$. We can write:

$$x_T(t) = A\text{rect}_{T_d}(t - T_d/2)\frac{1}{2j}(\exp(2j\pi F_0 t) - \exp(-2j\pi F_0 t))$$

If we use the modulation property (see page 727) satisfied by the Fourier transform of $\text{rect}_{T_d}(t - T_d/2)$, we get:

$$\begin{aligned} X_T(F) = \quad & A\frac{\sin(\pi(F-F_0)T_d)}{2j\pi(F-F_0)}e^{-j\pi(F-F_0)T_d} \\ & -A\frac{\sin(\pi(F+F_0)T_d)}{2j\pi(F+F_0)}e^{-j\pi(F+F_0)T_d} \end{aligned}$$

and therefore:

$$X_n = \frac{1}{T_d}X_T(n/T_d) = \frac{-2A}{\pi}\frac{1}{4n^2-1}$$

The continuous component is $X_0 = 2A/\pi$.

2. If we apply the Fourier transform to the differential equation, we get:

$$2j\pi RCFY(F) + Y(F) = X(F) \quad \Rightarrow \quad H(F) = \frac{1}{1 + 2j\pi RCF}$$

3. If the RC filter has a constant $RC \gg 1/F_0$ such that the components with the frequency $\pm 2kF_0$, where $k \geq 2$, are negligible, the continous component and the first harmonic are all that is left in the output. By remembering that the signal $\exp(2j\pi Ft)$, after filtering, leads to $H(F)\exp(2j\pi Ft)$, the output signal has the expression:

$$\begin{aligned} y(t) &\approx H(0)X_0 + H(-2F_0)X_{-1}e^{-4j\pi F_0 t} + H(2F_0)X_1 e^{4j\pi F_0 t} \\ &= \frac{2A}{\pi} + 2|H(2F_0)||X_1|\cos(4\pi F_0 t + \Phi) \end{aligned}$$

4. Because $T_e \ll RC$, approximating the derivative by:

$$\frac{\Delta y(t)}{\Delta t} = F_e(y_e(n) - y_e(n-1))$$

is acceptable. The differential equation becomes the recursive equation:

$$F_e(y_e(n) - y_e(n-1)) + \frac{1}{RC}y_e(n) = \frac{1}{RC}x_e(n)$$

Hence the simulation is equivalent to the filtering:

$$y_e(n)(1+\tau) - y_e(n-1) = \tau x_e(n)$$

if we choose $\tau = T_e/RC$.

5. The following program illustrates the cases of the half-wave and full-wave rectifiers (Figure H12.1):

```
%===== C2ALTERN.M
% Rectifiers
Fs=5000; Te=1/Fs; N=800; fa=50; A=220*sqrt(2);
t=(0:N-1)/Fs; xa=A*sin(2*pi*fa*t);
xrS=.5 *(sign(xa)+1) .* xa;      % Half-wave
xrD=abs(xa);                     % Full-wave
%===== Simulation of RC(dy/dt) + y = x
RC=.02; tau=Te/RC; mu=1/(1+tau); nu=tau*mu;
yS=filter(nu,[1 -mu],xrS); yD=filter(nu,[1 -mu],xrD);
subplot(211); plot(t,xrS,'-',t,yS,[0 max(t)],[A/pi A/pi]);
grid; subplot(212); plot(t,xrD,'-',t,yD,'-',...
              [0 max(t)],[(2*A)/pi (2*A)/pi]); grid
```

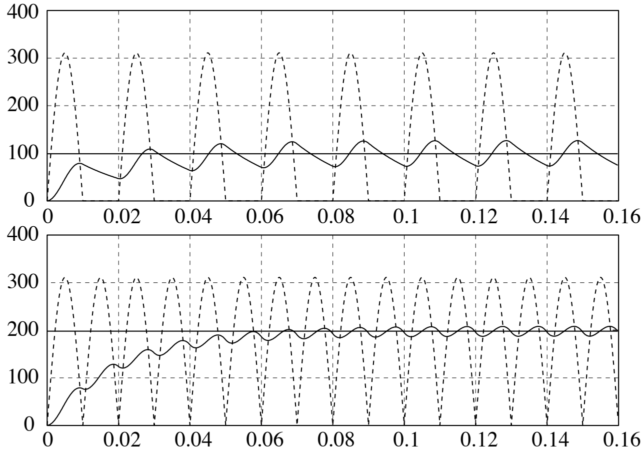


Figure H12.1 – *Simulation of the half-wave and full-wave rectifiers*

H12.2 (Simulation in the presence of a ZOH) (see page 454)

1. If we let $\mathbf{\Phi}(t) = e^{\mathbf{A}t}$ and notice that $e(u)$ remains constant between kT and $(k+1)T$, we have:

$$\begin{aligned}\mathbf{x}((k+1)T) &= \mathbf{\Phi}(T)\mathbf{x}(kT) + \int_{kT}^{(k+1)T} \mathbf{\Phi}((k+1)T-u)\mathbf{b}e(kT)du \\ &= \mathbf{\Phi}(T)\mathbf{x}(kT) + e(kT)\mathbf{\Psi}_{kT}\mathbf{b}\end{aligned}$$

   with:

$$\mathbf{\Psi}_{kT} = \int_{kT}^{(k+1)T} \mathbf{\Phi}((k+1)T-u)du = \int_0^T \mathbf{\Phi}(u)du = \mathbf{\Psi}(T)$$

   The output at the sampling times is given by $s(kT) = \mathbf{c}^T\mathbf{x}(kT)$.

2. The inter-sample response is given by:

$$\begin{cases} \mathbf{x}(t) = \mathbf{\Phi}(t-kT)\mathbf{x}(kT) + e(kT)\mathbf{\Psi}(t)\mathbf{b} \\ s(t) = \mathbf{c}^T\mathbf{x}(t) \end{cases}$$

   with $\mathbf{\Psi}(t) = \int_0^{t-kT}\mathbf{\Phi}(u)du$. In order to know the system's behavior between kT and $(k+1)T$, we need to know $\mathbf{x}(kT)$ and $\mathbf{\Psi}(t)$.

3. Type the following program:

```
%===== CREPETATS.M
% Sampling frequency (1/T)=10 Hz
% Simulation duration tmax=10 s
% Initial conditions x0
A=[0 1;-1 -1.4]; b=[0;1]; c=[1 0];
T=.1; tmax=10; x0=zeros(2,1);
[t,s]=Crepind(A,b,c,T,tmax,x0);
plot(t,s,'x'); grid
```

   The `Crepind` function computes the system's step response:

```
function [realt,xout] = Crepind(A,b,c,Ts,tmax,x0)
%%==================================================%
%% Step response of a linear system                 %
%% SYNOPSIS: [realt,xout]=CREPIND(A,b,c,Ts,tmax,x0) %
%% Entrees: (A,b,c) = State representation          %
%%          Ts      = Sampling frequency            %
%%          tmax    = Observation duration          %
%%          x0      = Initial state                 %
%% Sorties: realt   = Real time                     %
%%          xout    = Response                      %
```

```
%%==================================================%
npts=floor(tmax/Ts); [N,N]=size(A);  % System order
%===== Sampling frequency = 1/Ts Hz
Ae=[A b;zeros(1,N+1)]*Ts; Aexp=expm(Ae);
phi=Aexp(1:N,1:N); psib=Aexp(1:N,N+1);
tps=[0:npts-1]; realt=tps * Ts; xout=zeros(1,npts);
xx=x0;               % Initial conditions
xout(1)=c*xx;
for k=2:npts
    xx=phi * xx + psib; xout(k)=c * xx;
end
return
```

4. Changing from continuous-time over to discrete-time requires the use of the bilinear transform obtained with the `stoz.m` function from exercise 4.12:

```
%===== CETATSCOMP.M
% CREPETATS.M must be run before
tmax=10;
%===== Direct calculation  -> continuous time
A=[0 1;-1 -1.4]; b=[0;1]; c=[1 0]; x0=zeros(2,1);
[tps,s]=Crepind(A, b, c, T, tmax, x0);
%===== Calculating using the bilinear transform (T=0.1)
pol=[1 1.4 1]; T=.1; [DX,NX]=nbilin(pol,T);
N=floor(tmax / T); repind=filter(NX,DX,ones(1,N));
%===== Calculating using the bilinear transform (T=0.4)
T=.4; N=floor(tmax / T); [DX,NX]=nbilin(pol,T);
repind2=filter(NX,DX,ones(1,N));
tps2=[0:N-1] * T;
plot(tps,s,'-', tps,repind,'x', tps2,repind2,'o'); grid
```

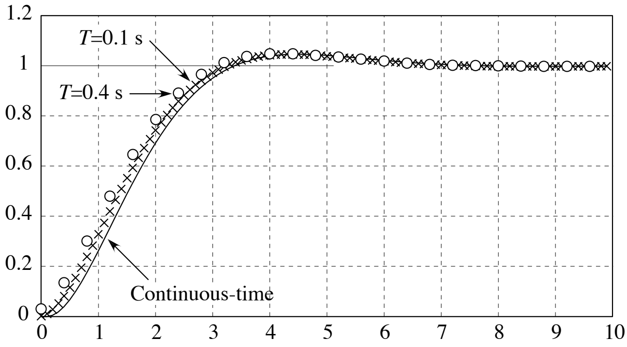


Figure H12.2 – *Comparing step response calculations: in continuous-time and in discrete-time using the bilinear transform*

H12.3 (Non-minimal system) (see page 455)

1. Simulation:

```
%===== REPETAT.M
clear
%===== System definition
atc=[-11/4 -11/8 -5/4;27/4 11/8 21/4;15/8 19/16 5/8];
btc=[1;-1;-1/2]; ctc=[3/8 1/2 -1/4];
%===== Parameters
T=.2;        % Sampling period
x0=[0;0;0];  % Initial state
tpm=130;     % Duration of the simulation
Npts=tpm/T; tps=[0:Npts]*T; e=ones(1,Npts);
%===== Discrete time equivalence
atd=expm(atc*T); abs(eig(atd))
btd=inv(atc)*(atd-eye(3,3))*btc;
ctd=ctc;
%===== Simulation
x=x0; s=[ctd*x0]; m=[max(max(abs(x0)))]; zz=[x0'];
for k=1:Npts
    x=atd*x+btd*e(k); s(k+1)=ctd*x;
    zz=[zz;x'];
    m(k+1)=max(max(abs(x)));
end
figure(1); %subplot(211); plot(tps,m); grid
title('Norm of the ste vector')
%subplot(212);
plot(tps,s,'-'); grid
```

2. The state vector's evolution indicates an instability.

3. The transfer function calculation ends up with:

$$G(s) = \mathbf{c}^T (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{b} = \frac{1}{s^2 + s + 1}$$

   An "unstable" pair is eliminated. From an input-output point of view, the system is stable. However, the presence of initial conditions causes the output to diverge because the unstable pole is not simplified in the free part of the response. It can be shown that the discrete time system has the same property.

4. The system diverges because of the computation noise (Figure H12.3). Calculating the roots of `poly(phi-psib*c')-poly(phi)` and of `poly(phi)` shows that there are a pole and a zero which are almost identical. As there is not really a simplification, this is another reason for the divergence of the output.

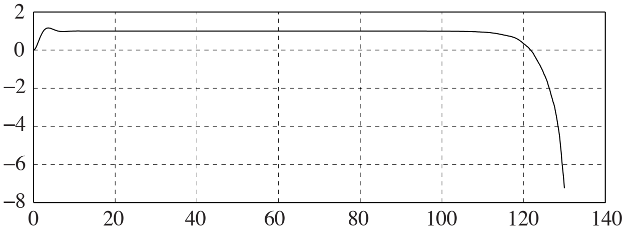


Figure H12.3 – *Response for a longer simulation time (*`tpm=130`*)*

H12.4 (Compression of a speech signal) (see page 469)

1. Pitch detection:

```
function [vnv,pitch]=detectpitch(sig,trhld,tmin,tmax,energm)
%%=======================================================%
%% Pitch detection using correlation                     %
%% SYNOPSIS:                                              %
%%     [vnv,pitch]=DETECTPITCH(sig,trhld,tmin,tmax,energm) %
%%    sig       = signal block                           %
%%    trhld     = correlation treshold                   %
%%    tmin,tmax = correlation window                     %
%%    energm    = energy threshold                       %
%%    vnv       = TRUE if voiced, otherwise FALSE        %
%%    pitch     = pitch period                           %
%%=======================================================%
nfa=length(sig); x=zeros(nfa,1); x(:)=sig; ae=x'*x;
if (ae > energm),              % energy>trhld
   for T=tmin:tmax
       stmT=x(T:nfa); s0T=x(1:nfa-T+1);
       autoc=stmT'*s0T; etmT=stmT'*stmT; e0T=s0T'*s0T;
       correl(T-tmin+1)=autoc/sqrt(etmT*e0T);
   end
   [corrmax,imax]=max(correl); tfond = imax+tmin-1;
   if (corrmax < trhld),
      vnv=(0==1); pitch=0; return;
   else
      pitch=tfond; vnv=(0==0);
   end
else
   pitch=-1; vnv=(0==1);
end;
return
```

Figure H12.4 shows the shape of the autocorrelation for a block. The maximum is located in $T = 82$, which means that the pitch frequency is roughly $f_0 = 8,000/82 \approx 97.5$ Hz. Determining T can become a difficult task when there are maxima present at the multiples of the pitch period. One solution is to check for the possible presence of a high maximum at sub-multiples of the x-coordinate found for the maximum. We can also study the evolution over different consecutive windows by comparing the obtained fundamental frequencies. Bear in mind, finally, that the accuracy can be improved by oversampling the signal beforehand.

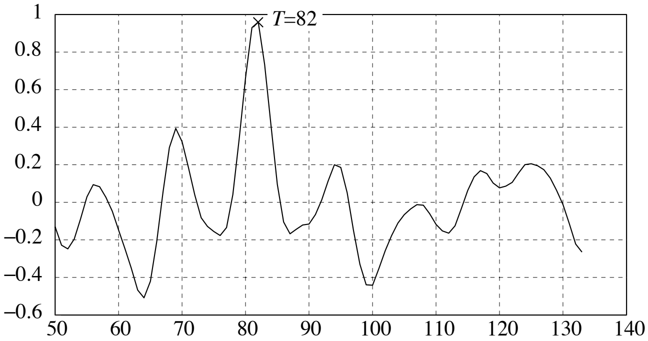


Figure H12.4 – *Examples of autocorrelation graphs*

2. Coding program (using the `xtoa` function from page 330):

```
%===== CODE.M
%%=======================================================%
%% code.m: Coding a speech signal based on an AR-model     %
%% INPUT:                                                  %
%%    Signal sampled at 8000 Hz                            %
%% OUTPUT: array tab_cod(N,XX):                            %
%%     tab_cod(N,1): energy in the block of signal         %
%%     tab_cod(N,2): pitch period                          %
%%     tab_cod(N,3:12): AR coefficients (AR-ordv if voiced %
%%         sound, AR-ordnv otherwise) or reflection coeffs. %
%% Each block has a 240 sample length (30 ms) with an      %
%% overlap of 60 samples.                                  %
%% Uses: xtoa        : AR-model coeffts                    %
%%       detectpitch : pitch detection                     %
%%       ai2ki       : reflection coeffs                   %
%%=======================================================%
clear
load phrase;              %===== Vector y
enerm=std(y)^2*.1;
% AR-model orders for voiced and non voiced sounds
ordv=20; ordnv=10;
NbParam=ordv+2;
phrase=y-mean(y);
%===== Parameters
```

```
lbloc=240;                             % Block length
recouv=60;                             % Overlap
ltr=lbloc-recouv;
nblocs=floor((length(phrase)-recouv)/ltr); % Nb of blocks
reste=rem(length(phrase)-recouv,ltr);
phrase=phrase(1:length(phrase)-reste);
tmin=40; tmax=150; seuil=0.7;          % For pitch detection
%=====
vnv=zeros(1,nblocs);           % Boolean "voiced/non voiced"
pitch=zeros(1,nblocs);         % Pitch period
tab_cod=zeros(nblocs,NbParam); % Coeffts of the model
%===== Detection "voiced/non voiced"
sprintf('"Voiced/non voiced" on %5.0f blocks', nblocs)
TIC
for k=1:nblocs,
    ind=(k-1)*ltr;
    blocan=phrase(ind+1:ind+lbloc);    % Analysis block
[vnv(k) pitch(k)]=detectpitch(blocan,seuil,tmin,tmax,enerm);
end;
TOC
%===== AR-model
sprintf('AR-model')
TIC
preacpar=filter([1 -0.9375],1,phrase); % Pre-emphasis
for k=2:(nblocs-1),
    %========
    if (vnv(k-1) == vnv(k+1)),   % Correction of
       vnv(k)=vnv(k-1);          % errors of detection
       if (vnv(k)==1)
          %==== "voiced" with pitch=mean
          pitch(k)=floor((pitch(k-1)+pitch(k+1))/2);
       else
          %==== "non voiced" with pitch=0
          pitch(k)=0;
       end
    end
    %===== Analysis block
    sigbloc=preacpar((k-1)*ltr+1:(k-1)*ltr+lbloc);
    if (vnv(k)==1)
       [pcoeff,enrg]=xtoa(sigbloc,ordv);     %<=======
       %=====> coeff_refl=ai2ki(pcoeff);     % Reflection
       %=====> tab_cod(k,3:NbParam)=coeff_refl; % coeffts
       tab_cod(k,3:NbParam)=pcoeff(2:ordv+1)';
       tab_cod(k,1)=enrg;
       tab_cod(k,2)=pitch(k);
    else
       [pcoeff, enrg]=xtoa(sigbloc,ordnv);
       %=====> coeff_refl=ai2ki(pcoeff);     % Reflection
       %=====> tab_cod(k,3:NbParam)= coeff_refl; % coeffts
       tab_cod(k,1)=enrg;
       tab_cod(k,2)=0;
       tab_cod(k,3:NbParam)=[pcoeff(2:ordnv+1)' ...
                             zeros(1,ordv-ordnv)];
    end;
end;
TOC
sprintf('Writing array in tab_cod.mat')
save tab_cod tab_cod
```

Notice the presence of a high-pass type pre-emphasis filter preceding the operations for estimating the model's parameters. We saw on page 652 a presentation of results for such an operation.

The prediction coefficients, obtained by analyzing the signal, are stored as "double floating point" numbers (8 bytes). If we had to use a "fixed-point" processor, it might be better to consider the reflection coefficients (refer to the Levinson algorithm on page 314) with values between -1 and $+1$. Speech coders also use what are called *lsp* coefficients, short for *Line Spectrum Pair* ([66]).

3. Decoding program:

```
%===== DECODE.M
% Decoding the file tab_cod.mat
clear
TIC
load tab_cod;                  % tab_cod(nblocs,XX);
excgl=eye(1,40);               % glottal signal
lbloc=240;                     % Block length
recouv=60;                     % Overlap
ltr=lbloc-recouv;
OvlRec=lbloc/3;                % Overlap reconstruction(1/3)
LBrec=lbloc+2*(OvlRec-recouv); % Reonstructed block length
nblocs=size(tab_cod,1); NbParam=size(tab_cod,2);
outsig=[]; finalsig=zeros(1,nblocs*ltr+OvlRec);
%===== Reconstruction window
fen_rec=[(1:OvlRec)/OvlRec ones(1,lbloc-2*recouv) ...
                              (OvlRec:-1:1)/OvlRec];
ImpGlPtr=0;
LgExcGl=length(excgl);
NbSmpTot=LBrec+ LgExcGl;       % Because of the filtering
drap_vnv=0;
%=====
for k=2:(nblocs-1),
    if (tab_cod(k,2)~=0)       %===== Voiced block
       if (drap_vnv==1)        % The previous one is voiced
          %==== Continuity of the input signal
          trame=[TmpSig(ltr+1:NbSmpTot), zeros(1,ltr)];
          NbSmp=NbSmpTot-ltr+ImpGlPtr;
       else              % The previous one is not voiced
          trame=zeros(1,NbSmpTot); NbSmp= 0;
       end
       PitchPeriod=tab_cod(k,2);    % Block pitch
       while (NbSmp<LBrec),
          trame((NbSmp+1):(NbSmp+LgExcGl))=excgl;
          NbSmp=NbSmp+PitchPeriod;
       end
       drap_vnv=1; ImpGlPtr=NbSmp-NbSmpTot;
       TmpSig=trame; trame=trame(1:LBrec);
       trame=trame/std(trame);        % Normalization
    else                              %===== Non voiced
       ImpGlPtr=0;
       drap_vnv=0;                    % Gaussian
       trame=randn(1,LBrec);          % white noise
    end;
    trame=sqrt(tab_cod(k,1))*trame;   % Power
    %den=ki2ai(tab_cod(k,3:NbParam)); %<===========
    den=[1 tab_cod(k,3:NbParam)];
    outsig=filter(1,den,trame); outsig=fen_rec .* outsig;
    st=(k-1)*ltr;
    %==== Construction with an overlap
    finalsig((st+1):(st+LBrec))=...
               finalsig((st+1):(st+LBrec)) + outsig;
end;
finalsig=filter(1,[1 -0.9375],finalsig);  % De-emphasis
TOC, soundsc(finalsig,8000);
```

The length of the block can be modified in the instruction `lbloc=240` of the `decode.m` program. By typing for example `lbloc=180`, the reconstructed windows are shorter and the same sentences are uttered faster. In both cases, this modification of the *utterance speed* occurs without a change in the timbre of the voice, which keeps its original, natural aspect. This is no longer the case if the sampling frequency is modified suddenly, with the same ratio by typing for example `soundsc(y,Fe*3/4)` to slow down the sentence. You can listen to the results and compare.

H12.5 (DTW) (see page 473)
Type the following function:

```
function [Dmin,DTWway,CD]=DTW1(xx,yy)
%%=============================================%
%% Synopsis: [Dmin,DTWway,CD]=DTW1(xx,yy)      %
%%      xx,yy  = cepstrum of x and y signals   %
%%      Dmin   = minimal cumulative distance    %
%%      DTWway = DTW way                       %
%%      CD     = array of cumulative distances %
%%=============================================%
```

```
mxx = 1; mxy = 2; myy = 1;
dd=size(xx,1); Ix=size(xx,2); Jy=size(yy,2);
distance=zeros(Ix,Jy);
for ix=1:Ix
    for jy=1:Jy
        diffe=xx(:,ix)-yy(:,jy);
        distance(ix,jy)=sqrt(diffe'*diffe);
    end
end
%===== Cumulative distance
CD = zeros(Ix, Jy);
%===== Parent to keep
Parent = zeros(Ix,Jy,2);
%===== CD initialization
CD(1,1) = distance(1,1);
Parent(1,1,:) = [1 1];
for ix = 2:Ix
    CD(ix,1) = distance(ix,1)+mxx*CD(ix-1,1);
    Parent(ix,1,:)= [ix-1 1];
end
for jy = 2:Jy
    CD(1,jy) = distance(1,jy)+myy*CD(1,jy-1);
    Parent(1,jy,:) = [1 jy-1];
end
%===== Main loop
nT = min(Ix, Jy);
for tt = 2:nT
    for ix = tt:Ix
        DD = [CD(ix-1,tt)+mxx*distance(ix,tt), ...
            CD(ix-1,tt-1)+mxy*distance(ix,tt), ...
            CD(ix,tt-1)+myy*distance(ix,tt)];
        [val,ind] = min(DD);
        CD(ix,tt) = val;
        switch ind
            case 1
                Parent(ix,tt,:)=[ix-1,tt];
            case 2
                Parent(ix,tt,:)=[ix-1,tt-1];
            case 3
                Parent(ix,tt,:)=[ix,tt-1];
        end
    end
    for jy = tt+1:Jy
        DD = [CD(tt-1,jy)+mxx*distance(tt,jy), ...
            CD(tt-1,jy-1)+mxy*distance(tt,jy), ...
            CD(tt,jy-1)+myy*distance(tt,jy)];
        [val,ind] = min(DD);
        CD(tt,jy) = val;
        switch ind
            case 1
                Parent(tt,jy,:)=[tt-1,jy];
            case 2
                Parent(tt,jy,:)=[tt-1,jy-1];
            case 3
                Parent(tt,jy,:)=[tt,jy-1];
        end
    end
end
%===== Normalization
%===== Minimal sum
Dmin = CD(Ix,Jy);
%===== Backtracking inverse
bi=zeros(nT+1,2);
%===== We start at the end
bi(1,:)=[Ix Jy];
ii=2;
while (Parent(bi(ii-1,1),bi(ii-1,2),1) ~= 1 ...
       & Parent(bi(ii-1,1),bi(ii-1,2),2) ~= 1)
    bi(ii,:) = Parent(bi(ii-1,1),bi(ii-1,2),:);
    ii = ii + 1;
end
DTWway = bi(ii-1:-1:1,:);
```

H12.6 (DTW word recognition) (see page 474)

1. Type the following function:

```
function cepstre=extractCEPSTRE(xt,Fe)
%%=========================================%
%% Synopsis: cepstre=EXTRACTCEPSTRE(xt,Fe) %
%%      xt      = Audio signal             %
%%      Fe      = Sampling frequency       %
%%      cepstre = Cepstral coefficients    %
%%=========================================%
xt=xt(:);
%===== Parameters
pp=10;                   % Cepstrum order
duree=15;                % Window duration in ms
Lfen=fix(Fe*duree/1000); % Window size
decal=fix(Lfen/2);       % Shift for overlapping
Lfft=2^nextpow2(Lfen);   % FFT size
hamm=0.5-0.5*cos(2*pi*(0:Lfen-1)'/Lfen);
Lx=length(xt);
nbfen=fix(Lx/decal);
cepstre = zeros(pp,nbfen);
for ii = 1:nbfen-1
    inddeb=(ii-1)*decal+1;indfin=inddeb+Lfen-1;
    xaux=xt(inddeb:indfin).*hamm;
    %===== Compute standard cepstral coefficients
    Sx = log(abs(fft(xaux,Lfft)));
    %===== Power cepstrum
    Cx = real(ifft(Sx));
    cepstre(:,ii)=Cx(2:pp+1); % Without energy
end
```

2. Type the following program:

```
%===== DTWTRY.m
clear all
[x,fe]=wavread('utter2.wav');
[y,fe]=wavread('utter4.wav');
cepx=extractCEPSTRE(x,fe);
cepy=extractCEPSTRE(y,fe);
[Dmin,wayDTW,CD]=DTW1(cepx,cepy); Dmin
figure(1); imagesc(CD');
set(gca,'ydir','normal')
hold on
plot(wayDTW(:,1),wayDTW(:,2),'k'); hold off
```

H12.7 (PSOLA) (see page 475)
Type the program:

```
%===== PSOLATRY.M
clear all;
[x,Fe]=wavread('desgens.wav');
gamma=0.8;
x_m=psola(x,Fe,gamma);
soundsc(x_m,Fe);
```

which calls the following function:

```
function s_synt=psola(s_orig,Fs,gamma)
%%=========================================%
%% SYNOPSIS: s_synt=PSOLA(s_orig,Fs,gamma) %
%%    s_orig    = Signal                   %
%%    Fs        = Sampling Frequency (Hz)  %
%%    gamma     = Modification rate        %
%%    s_synt    = Modified Signal          %
%% Uses the F0cor function                 %
%%=========================================%
seuil_pitch=0.7;
Rsurech=1;         % To improve the pitch's evaluation
L10ms=fix(Fs/100); % Constant size window (10 ms)
fp_min=70; fp_max=400; Lfen=2*fix(Fs/fp_min);
%=====
Ns=length(s_orig); Namax=fix(Ns*fp_max/Fs);
ta=zeros(Namax,1); ta(1)=1;
Pa=L10ms; inda=1;
%===== Analysis
```

```
while ta(inda)<Ns-Lfen
    indsdeb=ta(inda);
    % The length Lfen must be large enough
    % to allow the estimation of the lowest frequency
    indsfin=indsdeb+Lfen; sextrait=s_orig(indsdeb:indsfin);
    [Fpitch, corr]=...
       f0cor(sextrait,Fs,Rsurech,seuil_pitch,fp_min,fp_max);
    if isnan(Fpitch)
        Pa=L10ms;
    else
        Pa=fix(Fs/Fpitch);
    end;
    inda=inda+1; ta(inda)=ta(inda-1)+Pa;
end
ta=ta(1:inda);Na=length(ta);
%===== Time scale modification and Synthesis
s_synt=zeros(fix(Ns/gamma),1);
ii=1;ts=1;ie=1;te=1;
while ie<Na-2
    ii=ii+1;
    te=te+gamma; ie=ceil(te);
    Pa=ta(ie+1)-ta(ie); ts=ts+Pa;
    winHann=sin(pi*(0:2*Pa)'/(2*Pa)) .^2;
    sola=s_orig(ta(ie):ta(ie)+2*Pa) .* winHann;
    s_synt(ts-Pa:ts+Pa)= s_synt(ts-Pa:ts+Pa)+sola;
end
return
```

H12.8 (Hann window) (see page 477)
Type the following program:

```
%===== FENHANN.M
L=300; alpha=1/6;
n0=fix(alpha*L);
%===== Hann Window
hn=abs(sin(pi*(0:L-1)'/L)) .^2;
gn=hn.^2; pp=500; x=zeros(pp*n0,1);
for ii=0:pp-ceil(L/n0)
    id1=ii*n0;
    x(id1+1:id1+L)=x(id1+1:id1+L)+gn;
end
figure(1);plot(x)
max(x), sum(gn)/n0
%===== Plotting the DTFT of gn
figure(2); Lfft=4*1024; Gf=abs(fft(gn,Lfft));
plot((0:Lfft-1)/Lfft,20*log10(Gf))
set(gca,'xlim',[0 0.2]); hold on;
plot(ones(2,1)*(1:5)/n0,[-140*ones(1,5);40*ones(1,5)],':');
hold off
```

As you can see on the resulting graph, the sequence $x(n)$ is constant for any value L. The constant is equal to the sum of the elements of the sequence $g(n)$ divided by n_0. Therefore it depends on α. Notice that the property is still true for other powers of $h(n)$. By using the Poisson formula, we can show that this property is equivalent to the fact that the sequence $g(n)$ is equal to zero for the multiples of $1/n_0$. This amounts to choosing the inverse of an integer as the value of α and to choosing L so as to have $L\alpha$ equal to an integer.

H12.9 (Phase vocoder) (see page 478)
Type the program:

```
%===== PHCODER.M
clear all
[x,Fe]=wavread('desgens.wav');
gamma=0.8;
Lfft=256;
x_m = phasevoc(x, gamma, Lfft, Fe);
soundsc(x_m,Fe)
```

which uses the following functions:

```
function s_synt=phasevoc(s_orig,gamma,Lfft)
%%==============================================%
%% SYNOPSIS: s_synt=PHASEVOC(s_orig,gamma,Lfft) %
%%      s_orig = Audio source                   %
%%      gamma  = Modification rate              %
%%      Lfft   = FFT length                     %
%%      s_synt = modified audio signal          %
%% Uses tfct.m, specinterp.m and spec2sig.m     %
%===============================================%
% alpha is the shift rate relative to the FFT length
unsuralpha=8;     % Power of 2
n0=fix(Lfft/unsuralpha);
win=sin(pi*(0:Lfft-1)'/Lfft) .^2;
grandC=sum(win.^2)/n0;
%===== Initial STFT
spec_a = tfct(s_orig,Lfft,n0,win);
%===== Calculus of the modified DTFT
spec_s = specinterp(spec_a, gamma);
%===== Inversion
s_synt = spec2sig(spec_s, n0,win)/grandC;
return
```

```
function sig=spec2sig(spec,n0,win)
%%============================================%
%% Synthesis of a signal from its spectrogram %
%% SYNOPSIS: sig=SPEC2SIG(spec,ovlap)         %
%%    spec = Spectrogram                      %
%%    n0    = Shift value                      %
%%    sig   = Signal                           %
%%=============================================%
[Lfft,nbcol] = size(spec);
ispec=real(ifft(spec));
sig = zeros(Lfft+(nbcol-1)*n0,1);
%===== Re-synthesis using a window
for icol = 1:nbcol
    sigfen = ispec(:,icol) .* win;
    %===== Overlap-Add
    ixi = (icol-1)*n0+1;
    sig(ixi:ixi+Lfft-1) = sig(ixi:ixi+Lfft-1) + sigfen;
end
return
```

```
function spec_s=specinterp(spec_a, gamma)
%%===================================================%
%% Interpolation of a Short Term FT array            %
%% SYNOPSIS: spec_s=SPECINTERP(spec_a, gamma)        %
%%      spec_a = original spectrogram (Short Term FT) %
%%      gamma  = Temporal modification rate          %
%%      spec_s = modified spectrogram                %
%%===================================================%
[Lfft,nbcol] = size(spec_a);
ts=1:gamma:nbcol-1;
spec_s = zeros(Lfft,length(ts));
%===== Phase and Phase increase
phase_a = angle(spec_a);
module_a = abs(spec_a);
diffp = zeros(Lfft,1);
phase_s=phase_a(:,1);
indcol = 1;
for tt = ts
  %===== Two adjacent columns
  ta_min=floor(tt); ta_max=floor(tt)+1;
  %===== Weighted Mean
  pond = tt - floor(tt);
  modul = (1-pond)*module_a(:,ta_min) + pond *module_a(:,ta_max);
  spec_s(:,indcol) = modul .* exp(j*phase_s);
  %===== Phase diff and accumulation
  diffp = phase_a(:,ta_max)-phase_a(:,ta_min);
  phase_s = phase_s + diffp;
  indcol = indcol+1;
end
return
```

H12.10 (Spectral quantization noise shaping) (see page 481)

1. Type:

```
%===== CMISFQ1.M
%=====  Signal generation
surech=4; AA=[1.2 3.2 2.7];
freq=[437 504 1367]'/44100;
T=200; SE1=AA*cos(2*pi*freq*(0:T-1));
SE4=AA*cos(2*pi*freq*(0:surech*T-1)/surech);
xteff=std(SE1);
SE4int=interM(SE1,surech);
plage=(30:T-30);
SSBlim=20*log10(xteff/std(SE4(plage)- ...
            SE4int(plage+15)));
disp(sprintf('SNR of INTERM: %.2f',SSBlim));
```

Because the interM function only performs an approximation of the reconstruction formula, this operation introduces a calculation noise of about 60 dB. Hence in order for the quantization noise to be appreciable, its power has to be greater than the power of the calculation noise. If we choose for example a signal-to-quantization noise ratio equal to 48 dB, the quantization noise will be located about 12 dB above the calculation noise. This particular ratio corresponds (formula 7.34) to an 8 bit quantization. This is why, from now on, we will always set the number of bits below 8 when studying performances.

2. We saw in exercise 7.5 that the signal-to-quantization noise increases, theoretically, by 6 dB for every additional quantization bit. Hence going from 8 bits to 6 causes a loss of 12 dB. This is what we are going to check through simulation.

   In the suggested program, the peak value A_c of the quantization system is chosen equal to the maximum value determined for the entire test signal. In practice, this value can be obtained from an estimate P_x of the "instantaneous" power using an expression of the type $A_c = F\sqrt{P_x}$, where F refers to the clipping factor which usually has a value between 3 and 4. Type:

```
%===== CMISFQ2.M
clear; surech=4; AA=[1.2 3.2 2.7];
freq=[437 504 1367]'/44100;
T=200; SE1=AA*cos(2*pi*freq*(0:T-1));
SE4=AA*cos(2*pi*freq*(0:surech*T-1)/surech);
xteff=std(SE1);      % RMS value
Ac=max(abs(SE1));
%===== N1=8 bit quantization
nb1=8;               % Number of bits of the coder
q1=2*Ac/(2^nb1);     % Quantization step
SE1Q8=round(SE1/q1)*q1;
SSB_SE1Q8=20*log10(xteff/std(SE1-SE1Q8));
```

```
disp(sprintf('%i bits : SNR = %.2f',nb1,SSB_SE1Q8))
%===== N2=6 bit quantization
nb2=6; q2=2*Ac/(2^nb2); SE1Q6=round(SE1/q2)*q2;
SSB_SE1Q6=20*log10(xteff/std(SE1-SE1Q6));
disp(sprintf('%i bits : SNR = %.2f',nb2,SSB_SE1Q6))
```

3. Conducting this quantization operation on the oversampled signal allows us to distribute the total power of the quantization noise in the $(-F_{s2}/2, +F_{s2}/2)$ band.

   By going back to the $(-F_{s1}/2, +F_{s1}/2)$ band, that is to say the $(-1/8, 1/8)$ band in normalized frequencies (oversampling by a factor $M = 4$), the power of the quantization noise is divided by 4 whereas the useful signal's power remains the same, hence a gain of 6 dB. We can take this quantization of the oversampled signal to extremes, because the result depends on whether or not the quantization noise is white on the entire band. If this is not the case, the gain can be much smaller than 6 dB.

   After a 6 bit quantization of the oversampled signal, it is essential to filter it in order to eliminate the quantization noise outside the signal's useful band. This filtering is achieved by the 31 coefficient FIR filter with the cut-off frequency 1/8. The following program is an illustration of this result.

   Type:

```
%===== CMISFQ3.M
%===== Oversampling
SE4Q8=interM(SE1Q8,surech);    % Oversampling (8 bits)
%===== Truncation of SE4Q8 on 6 bits
SE4Q6HB=round(SE4Q8/q2)*q2; h=rif(31,1/(2*surech));
%===== Suppressing the out-of-band noise
SE4Q6=filter(h,1,SE4Q6HB);
%===== Avoiding side effects
bandm=(30:T-30);
SSB_SE4Q6=20* ...
   log10(xteff/std(SE4(plage)-SE4Q6(plage+30)));
disp(sprintf('%i bits, oversamp. rate %i',nb2,surech))
disp(sprintf('SNR = %.2f',SSB_SE4Q6))
```

4. We have (Figure H12.6) $z(n) = e(n) + u(n)$, $t(n) = z(n) - u(n)$ and $u(n) = y(n) - t(n-1)$. Therefore $z(n) = y(n) + \epsilon(n)$ where we have set:

$$\epsilon(n) = e(n) - e(n-1)$$

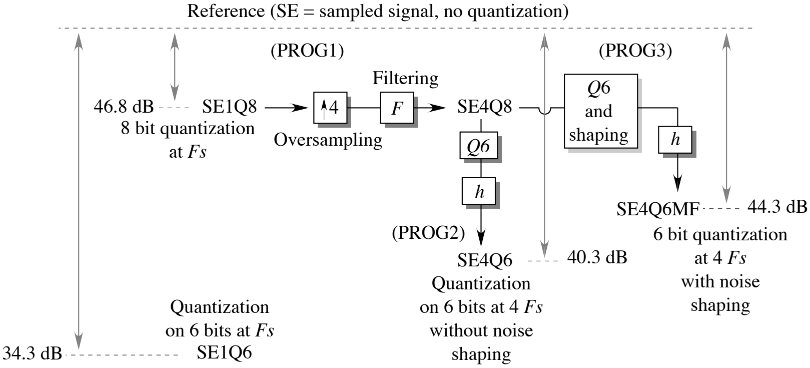

Reference (SE = sampled signal, no quantization)
(PROG1)
(PROG3)
Filtering
46.8 dB
SE1Q8
8 bit quantization at Fs
↑4
F
Oversampling
SE4Q8
Q6
h
Q6 and shaping
h
SE4Q6MF
44.3 dB
6 bit quantization at 4 Fs with noise shaping
(PROG2)
SE4Q6
40.3 dB
Quantization on 6 bits at 4 Fs without noise shaping
Quantization on 6 bits at Fs
34.3 dB
SE1Q6

Figure H12.5 – *Process simulations (experimental values)*

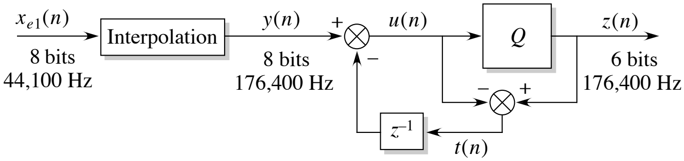


Figure H12.6 – *Noise shaping*

This causes an additive noise $\epsilon(n)$ to appear, obtained from the white noise $e(n)$, by a linear filtering with the impulse response $h(0) = 1$ and $h(1) = -1$. We can therefore calculate its power using the psd, the expression of which is given by $S_\epsilon(f) = |H(f)|^2 q^2/12$ where $|H(f)|^2$ represents the complex gain of the filter which is written:

$$H(f) = 1 - e^{-2j\pi f} = 2je^{-j\pi f}\sin(\pi f)$$

This means that $|H(f)|^2 = 4\sin^2(\pi f) = 2(1 - \cos(2\pi f))$. We have a high-pass filter. This quantization noise shaping makes it possible, by oversampling, to reduce the noise in the useful band (Figure H12.7).

If we oversample by a factor of 4, the useful signal ends up in the $(-1/8, 1/8)$ band. Therefore the power of the quantization noise has the value:

$$B = \frac{q^2}{12}\int_{-1/8}^{+1/8} 2(1 - \cos(2\pi f))df = \frac{q^2}{6}\left(\frac{1}{4} - \frac{\sin(\pi/4)}{\pi}\right) \approx 0.025\ \frac{q^2}{6}$$

Compared with the power of the previous noise, which was equal to $q^2/48$, the power of the noise is reduced by about -7 dB, which improves the

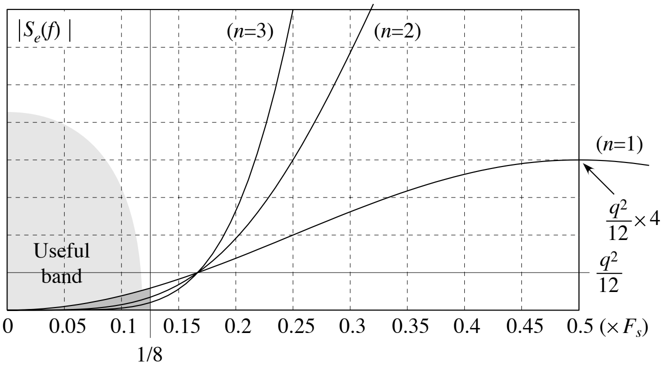


Figure H12.7 – *Effect of the the quantization noise shaping (dark gray), shown for* $n = 2$ *and* $n = 3$ *order differentiations*

signal-to-noise ratio by that much. However, this result is optimistic in the sense that the hypothesis of a white quantization noise is only partly true in practice. Type:

```
%===== CMISFQ4.M
%===== Noise shaping
euh=0;
for ii=1:length(SE4Q8)
    ut=SE4Q8(ii)+euh; zt(ii)=round(ut/q2)*q2;
    euh=ut-zt(ii);
end
SE4Q6NS=filter(h,1,zt);
SSB_SE4Q6NS=20* ...
    log10(xteff/std(SE4(bandm)-SE4Q6NS(bandm+30)));
disp(sprintf('SNRNS = %.2f',SSB_SE4Q6NS))
tt1=sprintf('%i bits ',nb2);
tt2=sprintf('Oversamp. rate %i with NS',surech);
disp([tt1 tt2])
disp(sprintf('SNR = %.2f',SSB_SE4Q6NS))
```

H12.11 (Denoising a speech signal) (see page 484)
Noising program:

```
%===== CNOISE.M
% Creation of the noisy file "phrasebruit"
T=1024; Lfft=4096; freq=(0:Lfft-1)/Lfft;
load phrase; % Speech signal file
ls=length(sn); nbi=fix(ls/T); ls=nbi*T; sn=sn(1:ls);
ps=sn'*sn/ls; SNR=10; SB=10 ^(SNR/10);
sigma=sqrt(ps/SB); sv1B=sn+sigma*randn(ls,1);
save phrasebruit sv1B sigma
```

Denoising program:

```
%===== CDENOISE2.M
%      PHRASEBRUIT = noisy signal
%      and sigma^2 = power of the white noise
clear; load phrasebruit;
N=256; Lfft=1024; lambda=1.9; mu=0.13;
seuil=lambda*sigma*sqrt(N); nbi=fix(length(sv1B)/N);
sv1deb=zeros(nbi*N,1); %== Denoised signal
for ii=0:nbi-1
    ind1=ii*N+1;ind2=ind1+N-1;
    xn=sv1B(ind1:ind2);
    Xk=fft(xn,Lfft); rap=seuil ./ abs(Xk);
    gain=(rap<1) .* (1-rap) + (rap>1) .* mu;
    Xkchap=gain .* Xk;
    xnchap=real(ifft(Xkchap,Lfft));
    sv1deb(ind1:ind2)=xnchap(1:N);
end
soundsc(sv1deb,8000)
```

λ and μ have been adjusted experimentally so as to achieve the best sound result. In practice, the psd of the noise is not known beforehand and has to be estimated based on the noisy data. The simplest method is to "visually" locate a signal segment containing nothing but noise and to estimate the psd in this segment. This method can effectively be applied to processes that do not have to be conducted in real-time, such as the restoration of recordings. In other cases, such as for example cellular technology, this operation has to be made automatic. But detecting a "silent" segment in the case of speech signals is no easy task, and the situation is even more complicated for music signals.

Figure H12.8 shows the effect of the frequency domain denoising operation.

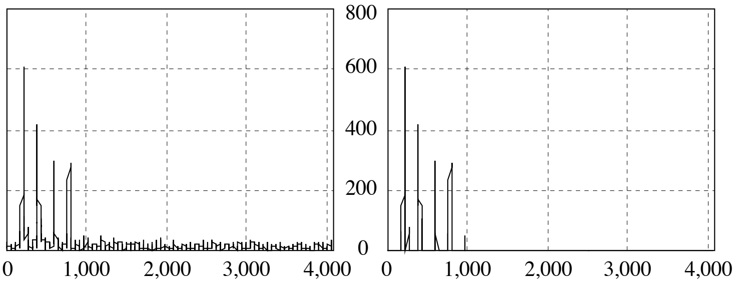


Figure H12.8 – *Spectra of the first block before (on the left) and after the denoising operation (on the right). The frequency is expressed in Hz*

H12.12 (Detecting impulse clicks) (see page 485)

1. We have:

$$\begin{aligned}\rho &= \frac{|z_d(n)|^2}{E\left(|z_b(n)|^2\right)} = \frac{\left|\sum_{u=-\infty}^{+\infty} g(u)d(n-u)\right|^2}{\sigma^2 \int_{-1/2}^{1/2} |G(f)|^2 df} \\ &= \frac{1}{\sigma^2} \frac{\left|\sum_{u=-\infty}^{+\infty} g(u)d(n-u)\right|^2}{\sum_{u=-\infty}^{+\infty} g^2(u)}\end{aligned}$$

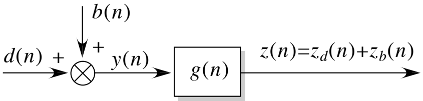


Figure H12.9 – *Matched filter*

2. If we apply the Schwarz inequality to the numerator, we get:

$$\left|\sum_{u=-\infty}^{+\infty} g(u)d(n-u)\right|^2 \le \sum_{u=-\infty}^{+\infty} g^2(u) \sum_{u=-\infty}^{+\infty} d^2(u)$$

and therefore $\rho \le E_d/\sigma^2$ where $E_d = \sum_{u=-\infty}^{+\infty} d^2(u)$. The resulting upper bound is reached if we assume $g(u) = d(n-u)$. It is therefore the maximum with respect to $g(u)$. Note that the optimal solution is the reversed copy of the signal $g(n)$. In the case where $d(u)$ has a finite duration k, we will assume $g(u) = d(k-u)$ in order for the filter $g(n)$ to be causal.

3. The filter with the transfer function $A(z) = 1 + a_1 z^{-1} + \cdots + a_K z^{-K}$ is a linear filter with the finite impulse response $\{h_1(n)\} = \{h_1(0) = 1, h_1(1) = a_1, \ldots, h_1(K) = a_K\}$. If we feed the signal $x(n) = \delta(n) + s(n)$ into this filter's input, we get the signal $y(n) = h_1(n) + w(n)$, which is the sum of the deterministic signal $h_1(n)$ and a white noise.

   If we apply the result of the previous question, the conclusion is that we have to filter the signal $y(n)$ by the filter with the impulse response $h_1(-n)$. Aside from a K sample delay, we get the *causal* filter with the impulse response $\{h_2(n)\} = \{h_2(0) = a_K, h_2(1) = a_{K-1}, \ldots, h_2(K) = 1\}$.

4. In the absence of clicks, the input signal $y(n)$ of the matched filter is a white noise with the variance σ^2. Hence the output signal is centered and the output spectral density has the expression $S(e^{2j\pi f}) = \sigma^2 |A(e^{2j\pi f})|^2$. The output power is obtained by integrating the spectral density. Using the Parseval formula, we get:

$$P_z = \sigma^2(1 + a_1^2 + \cdots + a_K^2) \tag{13.7}$$

5. In the absence of clicks, the output signal $z(n)$ of the matched filter $h_2(n)$ is a centered, Gaussian noise with the variance P_z. The probability of deciding the presence of a click is therefore given by:

$$\begin{aligned} \Pr(|z(n)| > s|H_0) &= 2\int_s^{+\infty} \frac{1}{\sqrt{2\pi P_z}} \exp(-u^2/2P_z)du \\ &= 2\int_{s/\sqrt{P_z}}^{+\infty} \frac{1}{\sqrt{2\pi}} \exp(-v^2/2)dv \\ &= 2Q(s/\sqrt{P_z}) \end{aligned}$$

where $Q(c)$ is the integral function of the centered, Gaussian distribution with the variance 1. If we choose $Q(c) = 0.005$, we have $c \approx 3$ and therefore:

$$s = 3\sqrt{P_z}$$

This threshold guarantees that the probability of deciding in favor of the presence of a click, when there is no click, is less than 1%: this is called the *probability of false alarm*. In order to set this level to satisfy a sound criterion, we have to compare the matched filter's output with a threshold of the type $\lambda\sqrt{P_z}$. The choice of λ will then be done by listening to the denoised signal. P_z can be estimated using expression 13.7.

6. The following program generates the useful signal comprising 500 samples of an AR-10. The impulses used to simulate clicks have an amplitude equal to 1.5 times the square deviation of the signal. The program then estimates the parameters, and computes the residual and the matched filter. Finally, the resulting signal is compared to a threshold (Figure H12.10):

```
%===== CCRAQ.M
clear
%===== Original signal (order 10-AR)
a= [1 -1.6507 0.6711 -0.1807 0.6130 -0.6085 0.3977 ...
    -0.6122 0.5412 0.1321 -0.2393];
K=length(a); N=500; w=randn(1,N);
```

```
s=filter(1,a,w); srms=sqrt(s*s'/N);
%===== NBCRAC clicks with an amplitude +/-1.5 srms
nbcrac=5; poscrac=[73 193 249 293 422];
ampcrac=1.5*srms*(2*round(rand(1,nbcrac))-1);
sig=s; sig(poscrac)=s(poscrac)+ampcrac;
subplot(311); plot(s); grid
subplot(312); plot(sig); grid
%===== Detection of the clicks
[aest sw2est]=xtoa(sig,K);  % Estimation of the AR
y=filter(aest,1,sig);       % Whitening: estim. of residual
z=filter(aest(K:-1:1),1,y); % Matched filtering
subplot(313); plot(z); grid
V0eff=sqrt(sw2est*aest'*aest);
lambda=3; threshold=lambda*V0eff;
izthreshold=find(abs(z)>threshold); % Threshold
izthreshold=izthreshold-K;          % Filter delay
lzs=length(izthreshold);
%===== Extraction of the maxima (3 samples from each other)
dist=izthreshold - [0 izthreshold(1:lzs-1)];
mpl3=find(dist>3); lm3=length(mpl3); mpl3=[mpl3 lzs+1];
for ii=1:lm3
    t1=izthreshold(mpl3(ii)); t2=izthreshold(mpl3(ii+1)-1);
    [zmax(ii) im]=max(z(t1:t2));
    posEstim(ii)=im+t1;
end
izthreshold,poscrac,posEstim
```

Note that the instruction **izthreshold=izthreshold-K**, which substracts K from the detected positions, takes into account the K sample delay caused by the causal implementation of the matched filter $h_2(n)$.

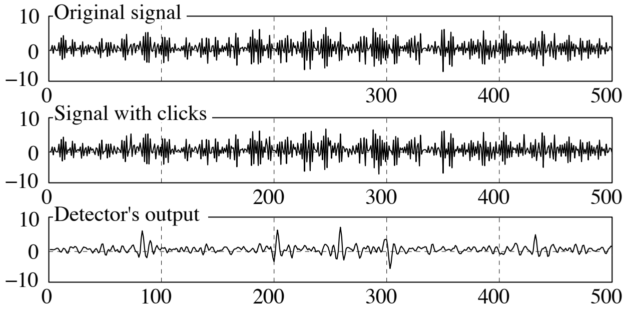


Figure H12.10 – *Signals obtained by declicking. The signal contains clicks that can clearly be located on the residual*

H12.13 (Restoring "missing values") (see page 488)

1. Let K be the order of the AR model and N the sample size. We wish to minimize the square deviation between the sequence of values $x(n)$ and the sequence of predicted values $\hat{x}(n) = -a_1 x(n-1) - \cdots - a_K x(n-K)$, for n from 1 to N, that is to say the quantity:

$$\sum_{u=1}^{N-K} (x(u) + a_1 x(u-1) + \cdots + a_K x(u-K))^2$$

The minimization is done with respect to m unknown values, with indices from ℓ to $\ell + m - 1$ (Figure H12.11).

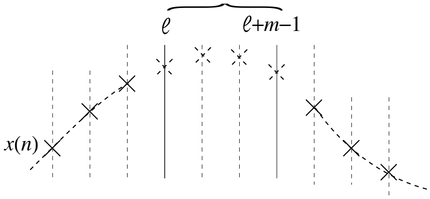


Figure H12.11 – *Several values are restored around the detected position*

These values are only involved in a limited number of terms of this sum which are:

$$J = \sum_{u=\ell}^{\ell+m+K-1} (x(u) + a_1 x(u-1) + \cdots + a_K x(n-K))^2$$

Notice that J appears as the norm of the vector:

$$\mathbf{e} = \mathbf{T}\mathbf{x}$$

where $\mathbf{T}$ is an $(m+K) \times (m+2K)$ Toeplitz matrix constructed from the coefficients $(a_1, \ldots, a_K)$:

$$\mathbf{T} = \begin{bmatrix} a_K & a_{K-1} & \cdots & a_1 & 1 & 0 & \cdots & \cdots \\ 0 & a_K & a_{K-1} & \cdots & a_1 & 1 & 0 & \cdots \\ \vdots & & & & & & & \end{bmatrix}$$

and $\mathbf{x}$ is a size $(m+2K)$ vector defined by:

$$\mathbf{x} = [\underbrace{x(\ell-K), \cdots, x(\ell-1)}_{\mathbf{x}_0\ :\ \text{known}}, \underbrace{x(\ell), \cdots, x(\ell+m-1)}_{\mathbf{y}\ :\ \text{unknown}}, \underbrace{x(\ell+m), \cdots, x(\ell+m+K-1)}_{\mathbf{x}_1\ :\ \text{known}}]^T$$

The size k vectors $\mathbf{x}_0$ and $\mathbf{x}_1$ are comprised of values that are known. The size m vector $\mathbf{y}$ is comprised of values we have to reconstruct.

Hence we can partition $\mathbf{T}$ in three matrices of the adequate size such that $\mathbf{e} = \mathbf{A}_0\mathbf{x}_0 + \mathbf{A}_1\mathbf{x}_1 + \mathbf{B}\mathbf{y}$, which we can also write $\mathbf{x} = \mathbf{A}_0\mathbf{x}_0 + \mathbf{A}_1\mathbf{x}_1 = -\mathbf{B}\mathbf{y} + \mathbf{e}$. In the end:

$$\mathbf{x} = -\mathbf{B}\mathbf{y} + \mathbf{e}$$

2. Minimizing the norm of $\mathbf{e}$ with respect to $\mathbf{y}$ involves the usual formulation of a least squares problem seen in chapter 11. The solution is:

$$\widehat{\mathbf{y}} = -(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{x} = -(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T(\mathbf{A}_0\mathbf{x}_0 + \mathbf{A}_1\mathbf{x}_1)$$

H12.14 (Contour ellipse: the least squares method) (see page 501)

1. Type:

```
%===== CONTOURELLIPSE.M
% Contour detection using a differentiation
% of the disk obtained with preprocesscoin.m
figure(2); subplot(221); imagesc(yim02);
colormap('gray'); axis('image'); title('Threshold')
%===== Center of the ellipse
yim0v=(Spix(1)-sum(yim02/255))/Spix(1);
nx=[1:Spix(2)]; mx=(yim0v * nx' / sum(yim0v));
hold on; plot([mx mx],[1 Spix(1)],'g');
yim0h=(Spix(2)-sum(yim02'/255))/Spix(2);
ny=[1:Spix(1)]; my=(yim0h * ny' / sum(yim0h));
plot([1 Spix(2)],[my my],'y'); hold off
%===== Digital differentiation
yim=yim02(5:145,5:295);
yshm=diff(yim); dimy=size(yshm);
subplot(222); imagesc(yshm);
colormap('gray'); axis('image')
title('Digital filter')
yims=ones(dimy)*255;
minsh=-220; maxsh=220;
ish1=find(yshm < minsh); yims(ish1)=zeros(size(ish1));
ish2=find(yshm > maxsh); yims(ish2)=zeros(size(ish2));
subplot(223); image(yims);
colormap('gray'); axis('image')
title('Threshold on the differentiation...')
%=====
[xm,ym]=find(yshm < minsh | yshm > maxsh);
subplot(224); plot((ym),(xm),'.')
set(gca,'Ydir','reverse'); axis('image')
```

```
title('Idem...')
%=====
figure(3)
pixc2=pixc(6:145,5:295); pixc2(ish1)=zeros(size(ish1));
pixc2(ish2)=zeros(size(ish2));
imagesc(pixc2);
colormap('gray'); axis('image'); title('Verification')
```

2. Theoretically the points of the ellipse obey the equation $ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 - 1 = 0$. Hence the idea of estimating the coefficients based on N pairs $\{x_1(n), x_2(n)\}$ by determining the value of $\boldsymbol{\theta} = [a \quad b \quad c \quad d \quad e]^T$ that minimizes:

$$(\mathbf{X}\boldsymbol{\theta} - \mathbf{u})^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{u})$$

where $\mathbf{X}$ is the $N \times 5$ matrix constructed from the sequences $x_1(n)$ and $x_2(n)$, and where $\mathbf{u}$ refers to the length N vector containing nothing but the components 1. The solution is given by:

$$\boldsymbol{\theta} = \mathbf{X}^{\#}\mathbf{u}$$

Once $\boldsymbol{\theta}$ has been estimated, we draw the ellipse using the `ellipse` function from exercise 21, which draws the ellipse with the equation $(\mathbf{x}-\mathbf{x}_0)^T\mathbf{E}(\mathbf{x}-\mathbf{x}_0) - \gamma = 0$. In order to do this, we have to determine the expressions which lead from $\boldsymbol{\theta}$ to the parameters $\mathbf{x}_0$, $\mathbf{E}$ and γ. By expanding $(\mathbf{x}-\mathbf{x}_0)^T\mathbf{E}(\mathbf{x}-\mathbf{x}_0)-\gamma = 0$, we get $e_{11}x_1^2+e_{22}x_2^2+2e_{12}x_1x_2-2\mathbf{x}^T\mathbf{E}\mathbf{x}_0+\mathbf{x}_0^T\mathbf{E}\mathbf{x}_0-\gamma = 0$ (e_{ij} refers to the generating element of $\mathbf{E}$ where $\mathbf{E} = \mathbf{E}^T$). If we identify this expansion as $ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 - 1 = 0$, we first have $e_{11} = a$, $e_{22} = b$ and $e_{12} = e_{21} = c/2$. Then, for any pair $\mathbf{x} = [x_1 \quad x_2]^T$, $-2\mathbf{x}^T\mathbf{E}\mathbf{x}_0 = dx_1 + ex_2 = \mathbf{x}^T[d \quad e]^T$, meaning that:

$$\mathbf{x}_0 = -\frac{1}{2}\mathbf{E}^{-1}\begin{bmatrix} d \\ e \end{bmatrix}$$

Finally, we have $\mathbf{x}_0^T\mathbf{E}\mathbf{x}_0 - \gamma = -1$. The following function draws the ellipse associated with the coefficients a, b, c, d and e.

Type:

```
%===== EQUATIONELLIPSE.M
undersamp=5;
%===== Reducing the number of points
[indz]=find(yims==0); Lindz=length(indz);
indz2=indz(1:undersamp:Lindz);
%=====
[indlig,indcol]=find(yims==0);
indcol2=indcol(1:undersamp:length(indcol));
indlig2=indlig(1:undersamp:length(indlig));
%===== Solution
x=indcol2; x2=x.*x; y=indlig2; y2=y.*y;
xy= x.*y; Xmat=[x2 y2 xy x y];
observ=[indcol2 indlig2];
%=====
bu=ones(length(indcol2),1);
coeff=Xmat\bu;
%=====
hold on; tracellipse(coeff); set(gca,'Ydir','reverse')
hold off
```

which uses the function `tracellipse`:

```
function tracellipse(coeff)
%%===================================================%
%% SYNOPSIS: TRACELLIPSE(coeff)                      %
%% coeff = Coefficients array                        %
%%    [coeff. of x^2, coeff. of y^2, coeff. of xy,...%
%%     coeff. of x, coeff. of y] corresponding to:   %
%%          ax^2+by^2+cxy+dx+ey-1=0                  %
%%===================================================%
a=coeff(1); b=coeff(2); c=coeff(3); d=coeff(4); e=coeff(5);
E(1,1)=a; E(2,2)=b; E(1,2)=c/2; E(2,1)=E(1,2);
X0=-inv(E)*[d;e]/2;
gam=X0'*E*X0 + 1; %gam=a*x0*x0+b*y0*y0+c*x0*y0+1;
N=100; theta=[0:N]*pi*2 / N;
Y=sqrt(gam)*[cos(theta);sin(theta)];
Fm1=inv(sqrtm(E)); X=diag(X0)*ones(2,N+1)+Fm1*Y;
plot(X(1,:),X(2,:),'r');
return
```

H12.15 (Contour ellipse: the covariance method) (see page 502)

1. Let U_1 and V_1 be the two components of $\mathbf{y}_1$. The mean vector $\mathbb{E}\{\mathbf{y}_1\}$ has two components. The first one is:

$$\mathbb{E}\{U_1\} = \int_{\mathbb{R}^2} u_1 p_Y(u_1, u_2) du_1 du_2 = \frac{1}{\pi} \int_{\mathcal{C}} u_1 du_1 du_2$$

If we assume $u_1 = \rho\cos(\theta)$, we get:

$$\mathbb{E}\{U_1\} = \frac{1}{\pi} \int_0^{2\pi} \int_0^1 \rho\cos(\theta)\rho d\rho d\theta = 0$$

We also have to check that the second component $\mathbb{E}\{V_1\} = 0$. In the end, $\mathbb{E}\{\mathbf{y}_1\} = 0$.

The covariance matrix $\mathbb{E}\{\mathbf{y}_1\mathbf{y}_1^T\}$ requires the calculation of three quantities $\mathbb{E}\{U_1^2\}$, $\mathbb{E}\{V_1^2\}$ and $\mathbb{E}\{U_1V_1\}$. If we assume $u_1 = \rho\cos(\theta)$, we get:

$$\mathbb{E}\{U_1^2\} = \int_{\mathbb{R}^2} u_1^2 p_{\underline{Y}}(u_1,u_2)du_1du_2 = \frac{1}{\pi}\int_0^{2\pi}\int_0^1 \rho^2\cos^2(\theta)\rho d\rho d\theta = \frac{1}{4}$$

Likewise, we have $\mathbb{E}\{V_1^2\} = 1/4$ et $\mathbb{E}\{U_1V_1\} = 0$. Therefore:

$$\mathbb{E}\{\mathbf{y}_1\mathbf{y}_1^T\} = \frac{1}{4}\mathbf{I}_2$$

2. The *law of large numbers* states that when N tends to infinity:

$$\frac{1}{N}\sum_{n=1}^{N}\left(\mathbf{y}_n - \hat{\boldsymbol{\nu}}_N\right)\left(\mathbf{y}_n - \hat{\boldsymbol{\nu}}_N\right)^T \xrightarrow{p.s.} \frac{1}{4}\mathbf{I}_2$$

If we multiply on the left and on the right by $\mathbf{M}^{1/2}$ and by remembering that according to 12.24, $\mathbf{M}\mathbf{y}_n = \mathbf{x}_n - \boldsymbol{\mu}$, we infer that:

$$\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_N)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_N)^T \xrightarrow{p.s.} \mathbf{M}/4$$

where $\hat{\boldsymbol{\mu}}_N = N^{-1}\sum_{n=1}^{N}\mathbf{x}_n$.

3. Type:

```
%===== RECHELLIPSECOVAR.M
clear all, close all
load piece2
[nblign nbcol]=size(pixc); [yy1,xx1]=find(pixc<70);
%=====
points=[xx1 yy1]; NN=length(xx1);
mymean=mean(points);
points_centres=points-ones(NN,1)*mymean;
RR=points_centres'*points_centres/NN;
image(pixc); hold on
MM=4*RR; E=inv(MM); ellipse(mymean,E,1)
hold off
```

H12.16 (Face recognition) (see page 512)

First, make a catalog of the data corresponding to the photographs in levels of gray. In our example, the catalog called `orlfaces` contains the catalogs called `s1`, `s2`, ..., each one containing the 10 photographs that have to be processed. Type the training program `LDAPCAtraining.m`. It is used by the test program `LDAPCAtest.m`:

```
%===== LDAPCATRAINING.M
k1=4; k2=3;
nbindiv=40;      % Number of individuals
nbimages=10;     % Number of photos
nbimages_A=3;    % Photos for training
XredPCA2D=cell(1,nbindiv);
XredLDA=cell(1,nbindiv);
V=cell(nbindiv,1);
W=cell(nbindiv,1);
for ii=1:nbindiv
    grandX=zeros(d,nbimages_A);
    grandXcell=cell(nbimages_A,1);
    for kimg=1:nbimages_A
        filename=sprintf([ImageFile '/s%i/%i.png'],ii,kimg);
        imge=imread(filename);
        grandXcell{kimg}=imge;
    end
    [V{ii}, W{ii}]=PCA2D(grandXcell,k1,k2);
    XredPCA2D{ii}=zeros(k1*k2,nbimages_A);
    for kimg=1:nbimages_A
        VTGW=V{ii}'*double(grandXcell{kimg})*W{ii};
        XredPCA2D{ii}(:,kimg)=reshape(VTGW,k1*k2,1);
    end
end
dim_barycenter=6;
gLDA=ALD(XredPCA2D,dim_barycenter);
barycenter_nua=zeros(dim_barycenter,nbindiv);
for ii=1:nbindiv
    XredLDA{ii}=gLDA'*XredPCA2D{ii};
    barycenter_nua(:,ii)=XredLDA{ii}*ones(nbimages_A,1)/nbimages_A;
end
```

Type the following recognition program:

```
%==== LDAPCATEST.M
clear all
ImageFile='orlfaces';
d=112*92;
nbindiv=40;      % Number of individuals
nbimages=10;     % Number of photos
nbimages_A=3;    % Photos for training
LDAPCAtraining;
matriceconf=zeros(nbindiv);
for ii=1:nbindiv
    for jj=nbimages_A+1:nbimages
        filename=sprintf([ImageFile '/s%i/%i.png'],ii,jj);
        grandXcell_T=double(imread(filename));
        AA=double(V{ii})'*grandXcell_T*double(W{ii});
        Xred_T=reshape(AA,k1*k2,1);
        XredLDA_T=gLDA'*Xred_T;
        for kk=1:nbindiv
            aux(kk)=norm(barycenter_nua(:,kk)-XredLDA_T,'fro');
        end
        [minaux indaux]=min(aux);
        matriceconf(ii,indaux)=matriceconf(ii,indaux)+1;
    end
end
matriceconf
```

H12.17 (Separating two sources) (see page 515)

1. We get $J(\alpha,\beta)=\sum_{\ell=1}^{L} J_\ell$ where:

$$J_\ell=\frac{\mathbf{C}_\ell^2(1,2)}{\mathbf{C}_\ell(1,1)\mathbf{C}_\ell(2,2)}$$

   with:

$$\begin{aligned}\mathbf{C}_\ell(1,1) &= r_{\ell,11}+2\alpha r_{\ell,12}+\alpha^2 r_{\ell,11}\\ \mathbf{C}_\ell(2,2) &= r_{\ell,11}+2\beta r_{\ell,12}+\beta^2 r_{\ell,11}\\ \mathbf{C}_\ell(1,2) &= (1+\alpha\beta)r_{\ell,12}+\beta r_{\ell,11}+\alpha r_{\ell,22}\end{aligned}$$

   where $r_{\ell,ij}$ are the 4 terms of the covariance matrix estimated on the ℓ-th block with the length N.

2. Type the following program:

```
%===== SEPARESOURCES.M
clear
load sigs      %===== s=[s1;s2]  array (2 x Ns)
fe=8000; Ns=size(s,2);
N=fe*10/1000;  %===== Block duration 10 ms
A=[1 1.3;-0.1 0.8];
%===== Mix of s1 and s2
x=A*s; y=zeros(Ns,2);
%===== N=window size
L=fix(Ns/N);
rell=zeros(2,2);
%===== Range for alpha and beta
alpha=(-2:0.01:-1);beta=(0:0.01:2);
la=length(alpha);lb=length(beta);
%===== JJ: evaluation function
JJ=zeros(la,lb);
for ii=1:L
    id1=(ii-1)*N+1;id2=id1+N-1;
    xell=x(:,id1:id2);
    xellc=xell-mean(xell')'*ones(1,N);
    R_ell=xellc*xellc'/N;
    r11=R_ell(1,1);r12=R_ell(1,2);r22=R_ell(2,2);
    for ia=1:la
        for ib=1:lb
            C12=(1+alpha(ia)*beta(ib))*r12+alpha(ia)*r22+...
                beta(ib)*r11;
            C11=r11+2*alpha(ia)*r12+alpha(ia)^2*r22;
            C22=r22+2*beta(ib)*r12+beta(ib)^2*r11;
            jell=C12^2/(C11*C22);
            JJ(ia,ib)=JJ(ia,ib)+jell;
        end
    end
end
%===== Minimum of JJ
[aa ida]=min(JJ); [aa idb]=min(aa);
alpha0=alpha(ida(idb)); beta0=beta(idb);
B=[1 alpha0;beta0 1]; y=B*x;
soundsc(y(1,:),fe); pause(3); soundsc(y(2,:),fe)
```

H12.18 (Radar telemetry) (see page 517)

1. In both cases, the energy is $\int_0^T |s_i(t)|^2 dt = A^2 T$.

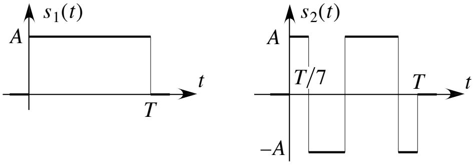


Figure H12.12 – *Two types of impulses*

2. In the absence of noise, the matched filter's output has the expression:

$$y(t) = \int s_i(u)h(t-u)du = \int s_i(u)s_i(u-t)du$$

The maximum is reached in $t = 0$ and is equal to $A_M = A^2 T$ in both cases.

3. Figure H12.13 shows the matched filter's input and output chronograms for the types of impulses that were chosen.

As you can see on the received signal, chronograms (a) and (c) in Figure H12.13, it is very difficult to decide the presence of an echo. That is the point of the matched filter. Notice, also, that the signal $y(t)$ is sharper when the impulse $s_2(t)$ is used, chronograms (b) and (d) in Figure H12.13. The fluctuations caused by the noise basically have the same amplitude in both cases. Therefore, the maximum is easier to locate when using the

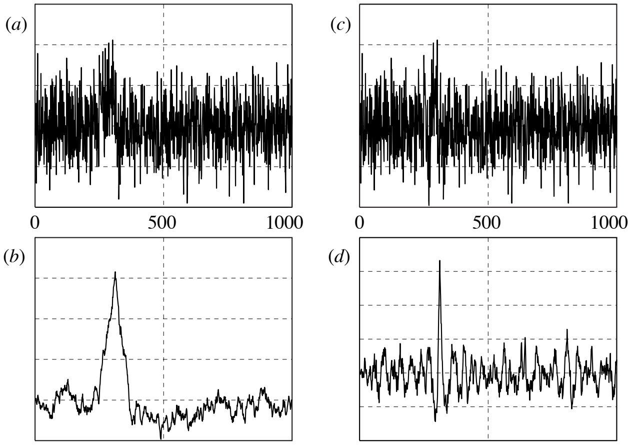


Figure H12.13 – *Matched filtering: (a) noisy signal $s_1(t)$, (b) matched filter's output for $s_1(t)$, (c) and (d) show the same information for the signal $s_2(t)$*

impulse $s_2(t)$. Because the two signals have the same energy, this means that performances are better for $s_2(t)$. Of course, this comes at the cost of a larger spectral area where $s_2(t)$ is different from zero. Type:

```
%===== CTELRAD1.M
Tmax=1000; T=63; Ts7=T/7; Es=T; trhld=Es/2; SNR=0;
sigmab=sqrt(10^(-SNR/10)); tau=200; trgt=(rand>.5);
if (trgt)
   disp(sprintf('Found target: tau=%g',tau));
   else
   disp('No target');
end;
s1=ones(T,1);
s2=[ones(Ts7,1);-ones(2*Ts7,1); ...
    ones(3*Ts7,1);-ones(Ts7,1)];
st1=[zeros(tau,1);s1;zeros(Tmax-T-tau,1)];
st2=[zeros(tau,1);s2;zeros(Tmax-T-tau,1)];
bt=sigmab*randn(Tmax,1);
xt1=trgt*st1+bt; xt2=trgt*st2+bt;
yt1=filter(s1(T:-1:1),1,xt1);
yt2=filter(s2(T:-1:1),1,xt2);
subplot(221); plot(xt1); subplot(223); plot(yt1)
subplot(222); plot(xt2); subplot(224); plot(yt2)
[ymax imax]=max(yt1);
if ymax>trhld
   tau1=imax-T; disp(sprintf('Pulse 1: tau1=%g',tau1));
   else
   disp('Pulse 1: no target')
```

```
end
[ymax imax]=max(yt2);
if ymax>trhld
   tau2=imax-T; disp(sprintf('Pulse 2: tau2=%g',tau2));
   else
   disp('Pulse 2 : no target')
end
```

The choice of the threshold is the result of a compromise between the probability of a false alarm and that of non-detection. If the threshold is increased, the probability of non-detection increases, while that of false-alarm decreases.

H12.19 (Denoising of an AR-1 signal using Kalman) (see page 523)

1. The state equation shows that the stationary solution is an AR-1 process. Therefore, its power has the expression:

$$\mathbb{E}\{x^2(0)\} = \frac{\sigma_b^2}{1-a^2}$$

2. According to 12.34, we have:

$$G(n) = \frac{K(n-1)}{K(n-1)+\sigma_u^2} \tag{13.8}$$

which leads us to $K(n-1)(1-G(n)) = \sigma_u^2 G(n)$. If we replace this result in 12.34, and notice that $\mathbf{C} = 1$ and that all the quantities are scalars, we successively have:

$$\begin{aligned} K(n) &= a^2K(n-1)(1-G(n))^2 + a^2G^2(n)\sigma_u^2 + \sigma_b^2 \\ &= a^2G(n)(1-G(n))\sigma_u^2 + a^2G^2(n)\sigma_u^2 + \sigma_b^2 \\ &= a^2\sigma_u^2 G(n) + \sigma_b^2 \end{aligned}$$

If we take $(n-1)$ and then replace $K(n-1) = a^2\sigma_u^2 G(n-1) + \sigma_b^2$ in the expression 13.8 that gives $G(n)$, we get the recursive formula:

$$G(n) = \frac{\rho + a^2G(n-1)}{1+\rho+a^2G(n-1)} \tag{13.9}$$

In this case, the initial conditions (see algorithm on page 519) lead to $K(0) = \mathbb{E}\{x^2(0)\} = \sigma_b^2/(1-a^2)$. Therefore $G(1) = \rho/(1+\rho-a^2)$. From the calculation point of view, everything happens as if we started out with formula 13.9 and the initial values $\hat{x}(0) = 0$ and $G(0) = \rho/(1-a^2)$.

The Kalman algorithm can be summed up as follows:

- Initial conditions: $\hat{x}(0) = 0$ et $G(0) = \rho/(1-a^2)$.
- For n from 1 to N:

$$\begin{cases} G(n) & = & \dfrac{\rho + a^2 G(n-1)}{1+\rho+a^2 G(n-1)} \\ \hat{x}(n) & = & a\hat{x}(n-1) + G(n)\left(y(n) - a\hat{x}(n-1)\right) \end{cases}$$

3. The following program is designed to test the algorithm:

```
%===== CKALM.M
N=200; mtime=(0:N-1); xch=zeros(N,1);
a=0.9;                   % Time constant for the AR-1
sigmab=1;                % Modelling noise with variance=1
%===== Trajectory
b=sigmab*randn(N,1); x=filter(1,[1 -a],b);
sigmau=3;                % Observation noise: variance=9
y=x+sigmau*randn(N,1); % Observation
saux=9*sigmau^2;
%===== Tracking
a2=a*a; rho=(sigmab/sigmau)^2; G(1)=rho/(1-a2);
for nn=2:N
    G(nn)=(rho+a2*G(nn-1))/(1+rho+a2*G(nn-1));
    xch(nn)=a*xch(nn-1)+G(nn-1)*(y(nn)-a*xch(nn-1));
end
plot(mtime,x,'-',mtime,y,':',mtime,xch,'o');
```

The results are shown in Figure H12.14.

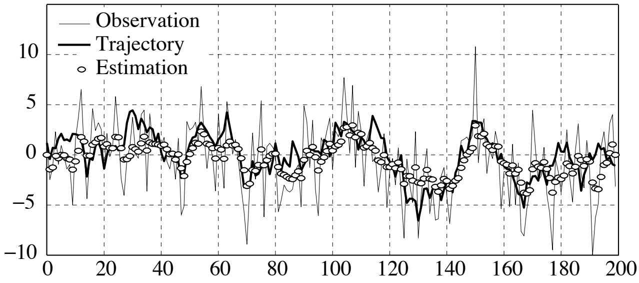


Figure H12.14 – *Results for the study of the filtering*

When we presented the Kalman filter, and implemented it in the previous program, we assumed that the model as well as the characteristic features of the noise were known. However, this is usually not the case. For example, if in our case the signal $x(n)$ is not an AR process, the choice of a and of σ_b^2 requires that we compromise between the ability of $x(n)$ to track the trajectory and the elimination of the noise. Choosing

a too close to 1 means that the model does not take into account the rapid variations of the signal $x(n)$. Therefore, the filter has difficulties "keeping up" with such variations. Likewise, if we choose σ_b^2 too high, we assume that we expect significant variations of the signal $x(n)$ with respect to the equation $x(n) = ax(n-1)$. You can check by using the previous algorithm and changing the parameters. Take for example `x=filter(1,[1 -a 0.2],)`.

H12.20 (Performances with two sub-codebooks) (see page 536)

1. Type:

```
%===== LBG64.M
N=5000; w=randn(2*N,1); xAR=filter(1,[1 0.9],w);
s=zeros(2,N); s(:)=xAR;
[CfD,CiD,ED]=lbg(s,64);
```

2. Type:

```
%===== LBGCOMP.M
% Comparison of the dictionaries
%===== First dictionary
[CfC1,CiC1,EC1]=lbg(s,4);
%===== Calculating the difference
diff=zeros(2,4*N);
for ii=1:4
   diff(:,(ii-1)*N+1:ii*N)=s-CfC1(:,ii)*ones(1,N);
end
%===== Second dictionary
[CfC2,CiC2,EC2]=lbg(diff,16);
```

In the first case, we obtained a mean square error `ED=0.1189`. In the second one, we obtained for the first (very crude) codebook `EC1=1.4684` and= for the second one, which is "more subtle", `EC2=0.5890`. Hence the second one gives the order of magnitude of the distortion caused by this second type of partition. We therefore have to compare, in terms of square deviation, the values 0.1189 and 0.5890. The loss is significant, but the total size of the two sub-codebooks is 20 instead of 64 for the maximum size dictionary.

H12.21 (Phase modulator) (see page 542)

1. Because the binary rate is equal to 1,500 bps, each bit lasts 1/1,500 s. Since the bits are arranged in groups of three, the transmission of each elementary signal lasts a duration of $T = 3 \times 1/1,500 = 2$ ms.

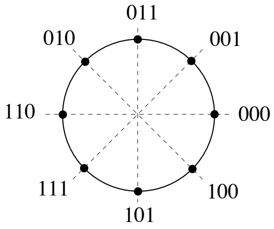


Figure H12.15 – *Gray code for an 8 state phase modulation*

2. Figure H12.15 shows a Gray code solution.

3. Type (see Figure H12.16):

```
%===== CPSK.M
M = 3;          % Bits per symbol
dpM = 2^M;      % Number of symbols
%===== Gray code
gray = [ 0 1 3 2 7 6 4 5 ];
% Exemple : 101=5 -> gray(5+1)=6
ak = exp(2*j*pi*gray/dpM);
%=====
Fe = 20000;     % Frequency for displaying the results
F0 = 2000;      % Carrier frequency
F0r = F0/Fe;
Db = 1500;      % Binary rate
T = M/Db;       % Symbol duration
NT = Fe*T;      % Number of display points
%===== 3K bits
seqbit = [0 0 0 1 0 1 0 0 1 1 0 0 0 1 1];
nbsymb=length(seqbit)/M;
seqgr = reshape(seqbit,M,nbsymb);
%===== Indices for the  symbols sequence
matcod = [4 2 1]; incod = matcod * seqgr + 1;
seqsymb = ak(incod);
%===== Complex envelope
ec = ones(NT,1)*seqsymb; ec = reshape(ec,1,nbsymb*NT);
tsa = (0:nbsymb*NT-1)/Fe;
sig = real(ec .* exp(2*j*pi*F0*tsa));
subplot(311); plot(tsa,real(ec)); grid
subplot(312); plot(tsa,imag(ec)); grid
subplot(313); plot(tsa,sig); grid
```

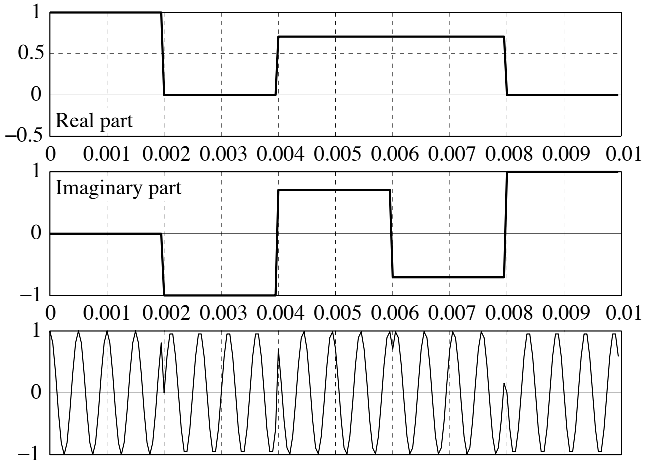


Figure H12.16 – *The real and imaginary parts of the complex envelope and the signal*

H12.22 (AMI code) (see page 547)

1. Because d_k is a uniform, i.i.d. sequence with possible values in $\{0,1\}$:

$$\mathbb{E}\{d_k\} = 1/2,\ \mathbb{E}\left\{d_k^2\right\} = 1/2 \text{ and } \mathbb{E}\{d_k d_j\} = \mathbb{E}\{d_k\}\,\mathbb{E}\{d_j\} = 1/4$$

for $k \neq j$. According to the coding rule:

$$\left\{\begin{array}{l} s_{k+1} = (1-2d_k)s_k \\ a_k = d_k\, s_k \end{array}\right.$$

therefore $a_k = d_k(1-2d_{k-1})(1-2d_{k-2})\dots$

Because the d_k are independent, $\mathbb{E}\{a_k\} = 0$. By expanding, we get $\mathbb{E}\left\{(1-2d_k)^2\right\} = 1$ and $\mathbb{E}\{(1-2d_{k-1})d_{k-1}\} = -1/2$.

Using the fact that the d_k are independent, we get $\mathbb{E}\left\{a_k^2\right\} = 1/2$ and:

$$\mathbb{E}\{a_k a_{k-1}\} = \mathbb{E}\{d_k\}\,\mathbb{E}\{(1-2d_{k-1})d_{k-1}\}\,\mathbb{E}\left\{(1-2d_{k-2})^2\right\},\dots$$

Therefore, $R_a(\pm 1) = -1/4$. We have to check that $\mathbb{E}\{a_k a_{k-j}\} = 0$ for $|j| \geq 2$. We infer that:

$$S_a(f) = -\frac{1}{4}e^{2j\pi fT} + \frac{1}{2} - \frac{1}{4}e^{-2j\pi fT} = \sin^2(\pi fT)$$

2. Type:

```
%===== AMI.M
N=10000; bits=(rand(1,N)>0.5); N=length(bits);
symb=zeros(1,N);
%===== Coder
vp=1;
for ii=1:N
    if (bits(ii)==1),
       symb(ii)=vp; vp=-vp;
       else symb(ii)=0;
    end;
end
srate=1000; Lfft=256; fq=srate*(0:Lfft-1)/Lfft;
blksz=50; ps=welch(symb,blksz,'rec',Lfft,0.95);
pt=sin(pi*fq/srate) .^2;
plot(fq,[ps pt']); grid; axis([0 srate/2 0 1.1]);
```

H12.23 (HDB3 code) (see page 547)

1. Starting with the initial values $p_v = +1$ and $p_1 = -1$, we have:

d_k	$\emptyset$	0	1	1	1	0	0	0	0	1	0	0	0	0	0	1	0
p_v	$+1$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	-1	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$+1$	$\cdots$	$\cdots$	$\cdots$
a_k	$\emptyset$	0	$+1$	-1	$+1$	-1	0	0	-1	$+1$	0	0	0	$+1$	0	-1	0
p_1	-1	$\cdots$	$+1$	-1	$+1$	$\cdots$	$\cdots$	$\cdots$	-1	$+1$	$\cdots$	$\cdots$	$\cdots$	$+1$	$\cdots$	-1	$\cdots$

2. Type:

```
function an=hdb3(dn,pv,p1)
%%=====================================================%
%% SYNOPSIS : an=HDB3(dn,pv,p1)                        %
%%    dn = Binary sequence to be coded                 %
%%    pv = Initial value of the bipolar violation bit %
%%    p1 = Initial value of the bipolar bit            %
%%    an = Coded sequence                              %
%%=====================================================%
N=length(dn); nz=0;
for ii=1:N
    if (dn(ii)==1)
        nz=0; an(ii)=-p1;p1=-p1;
    elseif (nz<3)
        nz=nz+1; an(ii)=0;
    else
        nz=0; an(ii)=-pv;
        if (pv==p1) an(ii-3)=-pv; end;
        pv=-pv; p1=pv;
    end
end
return
```

By typing:

```
hdb3([0 1 1 1 0 0 0 0 1 0 0 0 0 0 1 0],+1,-1)
```

check the result of the previous question.

3. Because the symbol sequence is centered in AMI coding, formula 12.54, which gives us the digital signal's spectrum, amounts to only the first term. The `welch` function then allows us to estimate the periodic part between 0 and $1/T$, corresponding to the correlations of the symbols a_n, that is to say:

$$S_a(f) = \sum_{\ell} R_a(\ell) e^{-2j\pi f \ell T}$$

at the frequency points $f = m/TL$ where $m = 0, \ldots, L-1$ and where L refers to the number of frequency points between 0 and $1/T$. All we have to do after that is multiply by the square modulus of the pulse spectrum, given, except for a multiplication factor (related to the amplitude), by $G(f) = \sin(\pi f T)/\pi f T$. If we restrict ourselves to the $(0, 1/T)$ frequency band, we have:

$$S_x(m/TL) = \frac{\sin(\pi m/L)}{\pi m/L} S_a(m/TL)$$

where the $S_a(m/TL)$ are estimated by applying the `welch` function to the sequence a_n. Type:

```
%===== TESTHDB3.M
clear; N=10000; bn=(rand(1,N)>0.5);
an=hdb3(bn,1,-1);
%===== Estimation of the spectrum of the sequence an
Lfft=256; fq=(0:Lfft-1)/Lfft; tblocs=200;
Saf=welch(an,tblocs,'rec',Lfft,0.95);
%===== Rectangular pulse on (0,T)
fq1=pi*(1:Lfft-1)'/Lfft;
Gf=[1;sin(fq1) ./ fq1]; Gf2=abs(Gf) .^2;
Sx= Saf .* Gf; plot(fq,Sx); grid
```

H12.24 (Linear equalization of a communications channel) (see page 549)

1. The constellation contains 16 symbols that can be coded using 4 bits. Remember that if F_0 refers to the carrier frequency, the transmitted signal can be written $s(t) = \mathrm{Re}(\alpha(t) e^{2j\pi F_0 t})$.

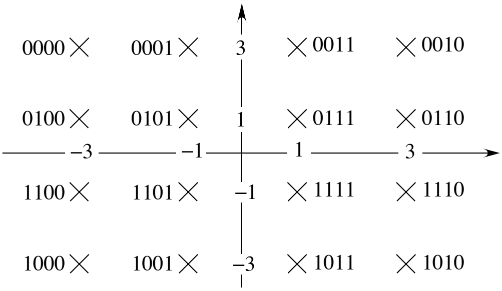


Figure H12.17 – *Constellation and Gray code*

2. The filter $G(z) = 1/(h_0+h_1z^{-1})$ is stable if the convergence area contains the unit circle. Therefore, if the square root of the denominator has a modulus greater than 1, the stable filter is anti-causal.

   For $h_0 = 1$ and $h_1 = -1.6$:

$$G(z) = \frac{1}{1-1.6z^{-1}} = -\frac{z}{1.6} - \frac{z^2}{1.6^2} - \frac{z^3}{1.6^3} - \cdots$$

   A length 21 causal approximation leads to a delay of 21 (see Figure H12.20).

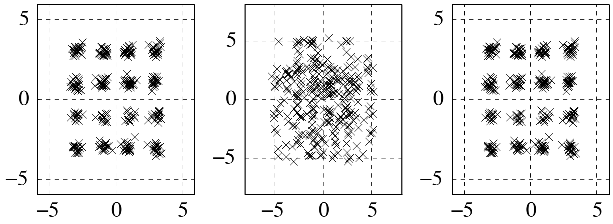


Figure H12.18 – *Noised signal, the signal-to-noise ratio is equal to* 20 *dB; signal containing the ISI caused by the FIR filter,* $h_0 = 1$ *and* $h_1 = -1.6$ *; signal after equalization*

3. Type:

```
%===== CMAQ.M
N=300; M=4;
%===== Alphabet
ar=2*(0:M-1)-M+1; un=ones(M,1); pa=2*ar*ar'/M;
ac=(un*ar+j*ar'*un'); M2=M^2; ind=ceil(M2*rand(N,1));
symb=zeros(N,1); symb(:)=ac(ind);
SNR=20; rho=10^(SNR/20); sb=sqrt(pa)/(rho*sqrt(2));
```

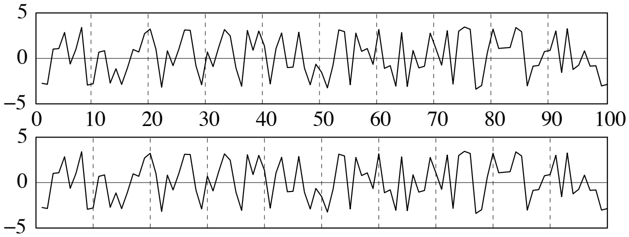


Figure H12.19 – *Comparison for a minimum phase channel*

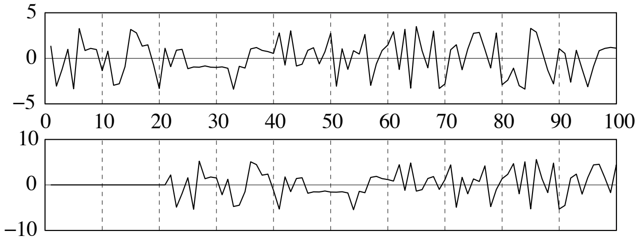


Figure H12.20 – *Comparison for a non-minimum phase channel*

```
br=sb*(randn(N,1)+j*randn(N,1)); xt=symb+br;
subplot(231); axis('square'); plot(xt,'x'); grid;
axis(1.2*[-1 1 -1 1]*max(abs(xt)));
%===== Non minimum phase  channel
% hc=[1 -1.6]; he=-(1/1.6) .^(20:-1:0);
%===== Minimum phase  channel
hc=[1 -0.6]; he=0.6 .^(0:20);
yt=filter(hc,1,xt); subplot(232); axis('square')
plot(yt,'x'); grid; axis(1.2*[-1 1 -1 1]*max(abs(yt)));
zt=filter(he,1,yt); subplot(233); axis('square')
plot(zt,'x'); grid; axis(1.2*[-1 1 -1 1]*max(abs(zt)));
subplot(413);plot(real(xt(1:100))); grid
subplot(414);plot(real(zt(1:100))); grid
```

H12.25 (2-PAM modulation) (see page 555)

1. Generating the coded signal:

```
%===== CMIA1.M
Fe=20000;      % Display frequency
Daff=500;      % Nb of display points
Db=1000;       % Binary rate = symbol rate
T=1/Db;        % Interval between symbols
```

```
NT=fix(Fe*T);
nbsymb=300; seqbits=round(rand(1,nbsymb));
seqsymb=2*seqbits-1; he=ones(NT,1); xe=he*seqsymb;
lx=NT*nbsymb; xe=reshape(xe,1,lx);   % NT pts/bit
tx=(0:lx-1)/Fe;
%===== Displaying only Daff points
subplot(221); plot(tx(1:Daff),xe(1:Daff)); grid
axis([tx(1) tx(Daff) -1.2 1.2]);
save miadata
```

2. Received signal:

```
%===== CMIA2.M
%      Output signal
clear; load miadata; hold off
lhsT=3.5;    % hc length
bc=0.06;     % Channel band
hc=rif(lhsT*NT-1,bc);
xr=filter(hc,1,xe);      % Filtering by the channel
subplot(222); plot(tx(1:Daff),xr(1:Daff))
axis([tx(1) tx(Daff) -1.3 1.3]); grid
save miadata
```

3. Type:

```
%===== CMIA3.M
% Eye pattern
clear; load miadata; hold off
%===== Matched filter output
h=conv(he,hc); lh=length(h); h=h/sum(h);
xa=filter(h(lh:-1:1),1,xr);
xo=zeros(2*NT,nbsymb/2);
%===== Length 2*NT  window (2 symbols)
for ii=1:nbsymb/2
    tdeb=(ii-1)*2*NT+1; tfin=tdeb+2*NT-1;
    xo(:,ii)=xa(tdeb:tfin)';
end
%===== Displaying the trajectories
subplot(223); plot((0:2*NT-1)/Fe,xo(:,5:nbsymb/2));
axis('square'); grid
%===== Choice of the decision time
disp('Choose the sampling time')
disp('corresponding to the widest aperture')
[aa bb]=ginput(1);
co=round(aa*Fe);          % Eye center
co=co-fix(co/NT)*NT       % between 0 and NT-1
save miadata
```

4. Type:

```
%===== CMIA4.M
% Displaying the result of sampling
clear; load miadata
hold off; subplot(224); axis('normal')
plot(tx(1:Daff),xa(1:Daff))
hold on; plot(tx(co+1:NT:lx),xa(co+1:NT:lx),'o')
hold off; axis([tx(1) tx(Daff) -1.3 1.3]); grid
```

5. Type:

```
%===== CMIA5.M
% Estimation of the error probability Pe
clear; load miadata
retard=ceil(lhsT)-round(co/NT)+1;
Px=xa*xa'/lx; SNR=1;
Pb=(NT/2) * Px/(10 ^(SNR/10));
zc=xr + sqrt(Pb)*randn(1,lx);
%===== Matched filtering
za=filter(h(lh:-1:1),1,zc(1:lx));
%===== Threshold detection
sbest=(sign(za(co+1:NT:lx))+1)/2;
sbest=sbest(retard:length(sbest));
seqbits=seqbits(1:length(sbest));
merr=find(sbest~=seqbits); nbe=length(merr);
pe=nbe/length(sbest);
disp(sprintf('SNR: %g dB',SNR))
disp(sprintf('Pe: %1.2f',pe))
```

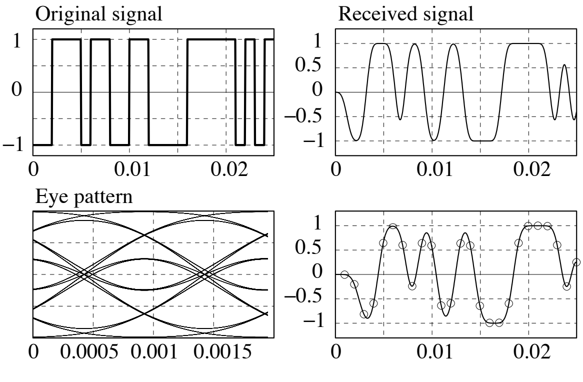


Figure H12.21 – *Signals at different points of the communication line*

H12.26 ("Zero Forcing" linear equalization) (see page 564)

1. If $g(n)$ refers to the channel's impulse response, the suggested equalizer has the impulse response $w(n)$, such that $w(n) \star g(n) = \delta(n)$. The equalizer's output signal therefore has the expression:

$$\begin{aligned} y(n) &= w(n) \star x(n) = w(n) \star g(n) \star a(n) + w(n) \star b(n) \\ &= a(n) + w(n) \star b(n) \end{aligned}$$

Notice that $y(n)$ only contains the contribution of the symbol $a(n)$. Interference due to other symbols is completely eliminated. This is why this equalizer is said to be "Zero Forcing", in the sense that it forces the ISI to be equal to zero.

2. The output noise $u(n) = w(n) \star b(n)$ of the filter $w(n)$ is Gaussian and centered. Using formula 8.63 for $P = 2$, we get the variance:

$$\sigma_u^2 = \mathbb{E}\{u^2(n)\} = \sigma^2 \frac{g_0 + g_2}{(g_0 - g_2)(g_0^2 + g_2^2 + 2g_0g_2 - g_1^2)}$$

3. We have $y(n) = a(n) + u(n)$. Therefore, under the hypothesis that $a_n = -1$, $y(n)$ is a Gaussian variable with the mean -1 and the variance σ_u^2. Under the hypothesis that $a_n = +1$, $y(n)$ is a Gaussian variable with the mean $+1$ and the variance σ_u^2.

4. The error probability is:

$$\begin{aligned} P_e &= \Pr(\text{Decide } -1 \text{ knowing } a_n = 1) \times \Pr(a_n = 1) \\ &\quad + \Pr(\text{Decide } 1 \text{ knowing } a_n = -1) \times \Pr(a_n = -1) \\ &= \Pr(y < 0 | a_n = +1)\frac{1}{2} + \Pr(y > 0 | a_n = -1)\frac{1}{2} \\ &= \frac{1}{2}\int_{-\infty}^{0} \frac{1}{\sigma_u\sqrt{2\pi}} \exp\left(-\frac{(y-1)^2}{2\sigma_u^2}\right) dy \\ &\quad +\frac{1}{2}\int_{0}^{+\infty} \frac{1}{\sigma_u\sqrt{2\pi}} \exp\left(-\frac{(y+1)^2}{2\sigma_u^2}\right) dy \\ &= \int_{\rho}^{+\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{v^2}{2}\right) dv = Q(\rho) \end{aligned}$$

with $\rho = 1/\sigma_u$. In MATLAB®, $Q(\rho)$ is obtained by typing `Q=(1-erf(rho/sqrt(2)))/2`.

5. Type:

```
%===== EGALLIN.M
clear
N=5000; g=[1 -1.4 0.8];
%===== Calculating sigma
Ch2=(g(1)+g(3))...
        /(g(1)^2+g(3)^2+2*g(1)*g(3)-g(2)^2)/(g(1)-g(3));
Ch=sqrt(Ch2);
%===== Several values for the SNR
SNRdB=(5:17); longSNR=length(SNRdB);
Pelin=zeros(longSNR,1); PeTheo=zeros(longSNR,1);
ak=sign(randn(1,N)); sk=filter(g,1,ak);
vs=sqrt(sk*sk'/N);
%=====
for jj=1:longSNR
    %=====
    RSB=10^(SNRdB(jj)/20); sigma_b=vs/RSB;
    bk=sigma_b*randn(1,N);
    %=====
    xk=sk+bk; ykegal=filter(1,g,xk);
    aklin=sign(ykegal(1:N));
    Pelin(jj)=sum(abs(ak-aklin)/2);
    %=====
    sigma_u=sigma_b*Ch;
    rho=1/sigma_u;
    %=====
    PeTheo(jj)=(1-erf(rho/sqrt(2)))/2;
end
Pelin=Pelin/N;
semilogy(SNRdB, Pelin, 'x', SNRdB, PeTheo, '-'); grid
```

Figure H12.22 shows the results. They are in perfect agreement with the theoretical values.

H12.27 (Wiener equalization) (see page 565)

1. Let us assume that $\mathbf{w} = (w(0), \cdots, w(N-1))^T$ and $\mathbf{x}(n) = (x(n), \ldots, x(n-N+1))^T$. The expression we have to minimize with respect to $\mathbf{w}$, is written $\mathbb{E}\left\{ |a(n-d) - \mathbf{w}^T\mathbf{x}(n)|^2 \right\}$. Using the projection principle, we have $a(n-d) - \mathbf{w}^T\mathbf{x}(n) \perp x(p)$ for $p \in \{n, \ldots, n-N+1\}$, which can be written $\mathbb{E}\left\{ (a(n-d) - \mathbf{w}^T\mathbf{x}(n))x(n-k)^* \right\} = 0$ where $k \in \{0, \ldots, N-1\}$, or also, in matrix form:

$$\mathbb{E}\{a(n-d)\mathbf{x}^*(n)\} = \mathbb{E}\left\{\mathbf{x}(n)\mathbf{x}^H(n)\right\}\mathbf{w} \tag{13.10}$$

2. If we assume that $a(n)$ is an identically and independently distributed sequence with possible values in $\{-1, +1\}$, we have $\mathbb{E}\{a_n\} = 0$ and:

$$R_{aa}(k) = \sigma_a^2\delta(k) \quad \text{avec} \quad \sigma_a^2 = 1$$

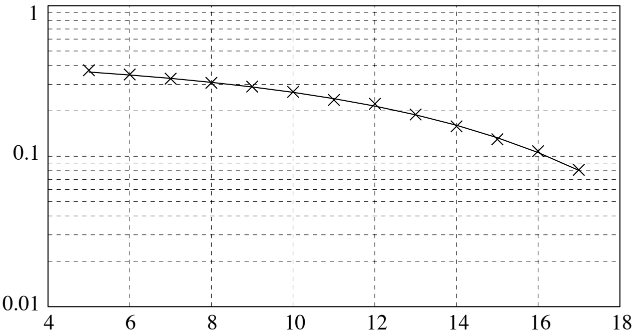


Figure H12.22 – *Symbol-by-symbol detection of a Zero Forcing equalizer's output for a binary transmission. The equivalent channel has the coefficients* $g_0 = 1$, $g_1 = -1.4$, $g_2 = 0.8$. *The* x*'s indicate the probabilities obtained through* 5,000 *simulations for different values of the signal-to-noise ratio in dB*

Let $s(n)$ be the channel output. According to the filtering formulas, we have:

$$R_{ss}(k) = g(k) \star g^*(-k) \star R_{aa}(k) = g(k) \star g^*(-k)$$

This sequence only has $2L - 1$ non-zero terms since $g(k)$ is of length L.

By assuming that $b(n)$ is independent of $a(n)$, $b(n)$ is independent of $s(n)$ and we have:

$$\begin{aligned} R_{xx}(k) &= \mathbb{E}\{(s(n+k) + b(n+k))(s^*(n) + b^*(n))\} \\ &= \mathbb{E}\{s(n+k)^* s(n)\} + \mathbb{E}\{b(n+k)b^*(n)\} \\ &= R_{ss}(k) + \sigma_b^2 \delta(k) \\ &= g(k) \star g^*(-k) + \sigma_b^2 \delta(k) \end{aligned}$$

Likewise, if we use the input/output filtering formula, we have:

$$\begin{aligned} R_{ax}(k) &= \mathbb{E}\{a(n+k)x^*(n)\} = \mathbb{E}\{a(n+k)(s^*(n) + b^*(n))\} \\ &= \mathbb{E}\{a(n+k)s^*(n)\} = g^*(-k) \end{aligned}$$

3. If we refer to expression 13.10, $\mathbb{E}\{\mathbf{x}(n)\mathbf{x}^H(n)\} = \mathbf{R}_{ss} + \sigma_b^2 \mathbf{I}$ where $\mathbf{R}_{ss}$ is a Toeplitz matrix constructed from the sequence $R_{ss}(k)$. The vector $\mathbb{E}\{a(n-d)\mathbf{x}^*(n)\}$ is comprised of the coefficients $g(k)$:

$$\mathbf{r}_{ax} = [\underbrace{0 \cdots 0}_{d} \underbrace{g^*(L-1) \cdots g^*(0)}_{L} \underbrace{0 \cdots 0}_{N-L-d}]^T$$

Therefore, the filter we are trying to determine is $\mathbf{w} = (\mathbf{R}_{ss} + \sigma_b^2 \mathbf{I}_N)^{-1} \mathbf{r}_{ax}$.

4. The **mmse.m** program allows us to examine the histograms of the received values as well as the histogram after equalization. As you can see, the intersymbol interference leads to a more spread-out histogram:

```
%===== MMSE.M
clear; N=5000; gc=[1 -1.4 0.8]; lg=length(gc);
ak=sign(randn(1,N)); % Sequence of symbols
sk=filter(gc,1,ak);  % Emitted signal
vsth=sqrt(gc*gc'); RSBdB=20;
%=====
sigma_b=vsth*10^(-RSBdB/20); bk=sigma_b*randn(1,N);
xk=sk+bk;            % Received signal
%===== MMSE (order-50 FIR)
LW=50; d=23;
rss=conv(gc,gc(lg:-1:1));
rsspos=[rss(lg:2*lg-1) zeros(1,LW-lg)];
Rxx=toeplitz(rsspos)+sigma_b*sigma_b*eye(LW);
ras=[zeros(1,d) gc(lg:-1:1) zeros(1,LW-lg-d)];
w=inv(Rxx)*ras';
ykwiener=filter(w,1,xk);
%====== SNR after equalization
roMMSE=max(conv(w,gc))/std(ykwiener);
%===== Displaying the results
points=50; subplot(211); hist(xk,points); grid
subplot(212); hist(ykwiener,points); grid
```

5. We can now perform the equalization with the Wiener filter, then, by simple detection with the threshold 0, estimate the binary sequence and measure the error probability. We can then compare the results for a Zero Forcing equalization. If the filter $g(n)$ is minimum phase, we can perform the Zero Forcing equalization simply by typing **filter(1,gc,xk)** and comparing the error probabilities. On the other hand, if the filter $g(n)$ is not minimum phase, the Zero Forcing equalization can only be achieved using the command **filter**, which implements the causal solution that is not stable. We then have to determine a causal and stable approximation of the inverse of $g(n)$.

   The following program implements the performance comparison in terms of error probability. After having run the **mmse.m** program, run the following program (Figure H12.23):

```
%===== EGALLINCMP.M
% Program to to be run after MMSE.M
SNRdB=(5:17); SNRlgth=length(SNRdB); retardW=d+lg-1;
%=====
for jj=1:SNRlgth
    RSB=10^(SNRdB(jj)/20); sigma_b=vsth/RSB;
    bk=sigma_b*randn(1,N); xk=sk+bk;
    ykegalZF=filter(1,gc,xk); aklinZF=sign(ykegalZF(1:N));
    PelinZF(jj)=sum(abs(ak-aklinZF)/2);
    %=====
    Rxx=toeplitz(rsspos)+sigma_b*sigma_b*eye(LW);
    w=inv(Rxx)*ras'; ykegalW=filter(w,1,xk);
    aklinW=sign(ykegalW(1:N));
PelinW(jj)=sum(abs(ak(1:N-retardW)-aklinW(retardW+1:N))/2);
end
PelinZF=PelinZF/N; PelinW=PelinW/N;
semilogy(SNRdB, PelinZF,'x'); hold on;
semilogy(SNRdB,PelinW,'o'); hold off; grid
```

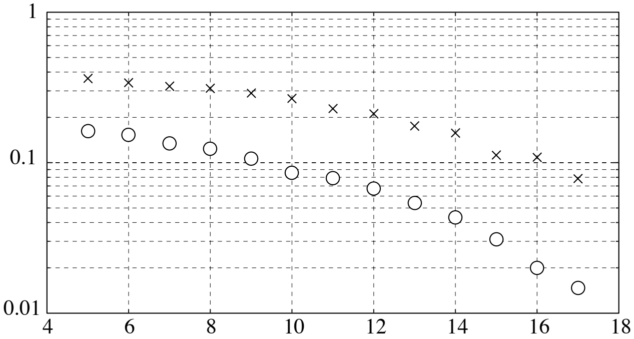


Figure H12.23 – *Error probability plotted against the signal-to-noise ratio in dB after equalizing with the Wiener filter ('*○*') and with the Zero Forcing filter ('*×*'). The results are obtained through simulation using* 5,000 *symbols. The channel filter has the finite impulse response* (1 $-$1.4 0.8)

Notice that the results are significantly better with the Wiener filter.

This page intentionally left blank

Chapter 14

Appendix

A1 Fourier transform

Property 14.1 *The main properties of the DFT are listed below:*

- *$X(f)$ is bounded, continuous, tends towards 0 at infinity and belongs to $L_2(\mathbb{R})$;*
- *the Fourier transform is linear;*
- *expansion/compression of time: the Fourier transform of $x(at)$ is $\frac{1}{|a|}X(f/a)$;*
- *delay: the Fourier transform of $x(t-t_0)$ is $X(f)e^{-2j\pi f t_0}$;*
- *modulation: the Fourier transform of $x(t)e^{2j\pi f_0 t}$ is $X(f-f_0)$;*
- *conjugation: the Fourier transform of $x^*(t)$ is $X^*(-f)$. Therefore, if the signal $x(t)$ is real, $X(f) = X^*(-f)$. This property is said to be of* hermitian symmetry*;*
- *if the signal $x(t)$ is real and even, $X(f)$ is real and even;*
- *if the signal is purely imaginary and odd, $X(f)$ is purely imaginary and odd;*
- *the convolution product, written $(x \star y)(t)$, is defined by:*

$$(x \star y)(t) = \int_{-\infty}^{+\infty} x(u)y(t-u)du = \int_{-\infty}^{+\infty} x(t-u)y(u)du \qquad (14.1)$$

  and has $X(f)Y(f)$ as its Fourier transform;

- *likewise, the Fourier transform of $x(t)y(t)$ is $(X \star Y)(f)$;*

- *if $x(t)$ is m times continuously differentiable and if its derivatives are summable up to the m-th order, then the Fourier transform of the m-th derivative $x^{(m)}(t)$ is $(2j\pi f)^m X(f)$;*
- *if $t^m x(t)$ is summable, then the Fourier transform of $(-2j\pi t)^m x(t)$ is the m-th derivative $X^{(m)}(f)$.*

A2 Discrete time Fourier transform

Property 14.2 *Let $X(f)$ and $Y(f)$ be the DTFTs of the sequences $\{x(n)\}$ and $\{y(n)\}$ respectively. The DTFT has the following properties:*

1. *linearity: $ax(n) + by(n) \rightarrow aX(f) + bY(f)$;*
2. *time-shift:*
$$x(n - n_0) \rightarrow X(f)e^{-2j\pi n_0 f} \tag{14.2}$$
3. *modulation: $x(n)e^{2j\pi f_0 n} \rightarrow X(f - f_0)$;*
4. *time reversal: $x(-n) \rightarrow X(-f)$;*
5. *conjugation: $x^*(n) \rightarrow X^*(-f)$;*
6. *real sequence: $x(n)$ real $\rightarrow X(f) = X^*(-f)$. $X(f)$ has a property called hermitian symmetry. Particularly, $|X(f)|$ and the real part $\mathrm{Re}(X(f))$ are even functions. Its phase $\arg(X(f))$ and its imaginary part $\mathrm{Im}(X(f))$ are odd functions. In this case, the plotting of $X(f)$ can be limited to the interval $f \in (0, 1/2)$.*
7. *convolution: the* convolution product *or* convolution *defined by:*
$$x(n) \star y(n) = \sum_{k=-\infty}^{+\infty} x(k)y(n-k) = \sum_{k=-\infty}^{+\infty} x(n-k)y(k)$$
has the product $X(f)Y(f)$ as its DTFT.

As an exercise, we will now demonstrate part 6 of properties 14.2. Starting off with the definition 2.21, we get, after conjugating and changing the sign:

$$X^*(-f) = \sum_{n=-\infty}^{+\infty} x^*(n)\exp(-2j\pi nf)$$

which is still equal to $X(f)$ since $x^*(n) = x(n)$.

A3 Discrete Fourier transform

Property 14.3 *The main properties of the DFT are listed below:*

1. *linearity:* $ax(n) + by(n) \to aX(k) + bY(k)$*;*
2. *time-shift:* $x((n-p) \bmod N) \to X(k)e^{-2j\pi pk/N}$*;*
3. *time reversal:* $x((-n) \bmod N) \to X((-k) \bmod N)$*;*
4. *conjugation:* $x^*(n) \to X^*((-k) \bmod N)$*;*
5. *real sequence:* $x(n)$ *real* $\to X(k) = X^*((-k) \bmod N)$*;*
6. *circular convolution: the sequence* $Z(k) = X(k)Y(k)$ *has:*

$$z(n) = \sum_{p=0}^{N-1} x(p)y((n-p) \bmod N) \tag{14.3}$$

   as its inverse DFT.

7. *Parseval formula:*

$$\sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N}\sum_{k=0}^{N-1} |X(k)|^2 \tag{14.4}$$

• As an example, consider the case of time reversal (item 3) for $N = 4$. The sequence $y(n) = x(-n)$ refers to:

$$\{y(0), y(1), y(2), y(3)\} = \{x(0), x(3), x(2), x(1)\}$$

Its DFT is:

$$\{Y(0), Y(1), Y(2), Y(3)\} = \{X(0), X(3), X(2), X(1)\}$$

• To prove expression 14.3 of item 6, we will calculate the inverse DFT of $Z(k) = X(k)Y(k)$, using 2.32. We get:

$$\begin{aligned}
z(n) &= \frac{1}{N}\sum_{k=0}^{N-1} X(k)Y(k)e^{2j\pi nk/N} \\
&= \frac{1}{N}\sum_{k=0}^{N-1}\left(\sum_{\alpha=0}^{N-1} x(\alpha)e^{-2j\pi\alpha k/N}\sum_{\beta=0}^{N-1} y(\beta)e^{-2j\pi\beta k/N}\right)e^{2j\pi nk/N} \\
&= \sum_{\alpha=0}^{N-1}\sum_{\beta=0}^{N-1} x(\alpha)y(\beta)\left(\frac{1}{N}\sum_{k=0}^{N-1} e^{-2j\pi(\alpha+\beta-n)k/N}\right)
\end{aligned}$$

The equality 2.33 leads us to the expected result.

- To prove item 7, all you have to do is set $y(n) = x^*(-n)$ in 14.3, then calculate $z(n)$ for $n = 0$.

A4 z-Transform

Property 14.4 *Let $X_1(z)$ and $X_2(z)$ be the z-transforms of the sequences $\{x_1(n)\}$ and $\{x_2(n)\}$ respectively, and $\mathcal{D}_1$ and $\mathcal{D}_2$ their convergence areas. We have the following properties:*

1. *Linearity:*

$$a_1x_1(n) + a_2x_2(n) \to a_1X_1(z) + a_2X_2(z) \tag{14.5}$$

with $\mathcal{D} = \mathcal{D}_1 \cap \mathcal{D}_2$.

2. *Time delay:*

$$x(n-k) \to z^{-k}X(z) \tag{14.6}$$

The convergence area is unchanged.

3. *Time reversal:*

$$x(-n) \to X(1/z) \quad \textit{with} \quad \frac{1}{R_2} < |z| < \frac{1}{R_1} \tag{14.7}$$

4. *Reality and symmetry: if the sequence $\{x(n)\}$ is real, then:*

$$X(z) = X^*(z^*) \tag{14.8}$$

The convergence area is unchanged.

5. *Convolution:*

$$(x_1 \star x_2)(n) = \sum_{k=-\infty}^{+\infty} x_1(n-k)x_2(k) \to X_1(z)X_2(z) \tag{14.9}$$

with $\mathcal{D} \supset \mathcal{D}_1 \cap \mathcal{D}_2$.

6. *Parseval relation:*

$$\sum_{n=-\infty}^{+\infty} |x(n)|^2 = \frac{1}{2j\pi} \oint_{(\Gamma)} X(z)X^*(1/z^*)\frac{dz}{z} \tag{14.10}$$

where the integration contour (Γ) is inside the convergence areas of $X(z)$ and of $X^*(1/z^*)$. Note that the unit circle is inside the convergence area.

7. *Any function $X_z(z)$, holomorphic inside the ring $R_1 < |z| < R_2$, is expandable in a* unique *power series $X_z(z) = \sum_{n\in\mathbb{Z}} x(n)z^{-n}$ with:*

$$x(n) = \frac{1}{2j\pi} \oint_{(\Gamma)} X(z)z^{n-1}dz$$

where (Γ) refers to a Cauchy contour [27] inside the ring $R_1 < |z| < R_2$. This integral can be calculated with the use of Cauchy's integral formula (also known as the "residue method").

8. *The sum of the power series $X_z(z) = \sum_{n\in\mathbb{Z}} x(n)z^{-n}$ is a* holomorphic *function in the convergence area $R_1 < |z| < R_2$ and its derivative can be obtained term-by-term:*

$$nx(n) \to -z\frac{dX_z(z)}{dz} \tag{14.11}$$

The convergence area is unchanged.

The results below show that the convergence area's shape is related to the properties of the sequence $\{x(n)\}$.

Property 14.5 *We have the following:*

1. *$x(n)$ is such that $\sum_n |x(n)| < +\infty$ if and only if the unit circle belongs to the convergence area.*

2. *The signal is causal if and only if the convergence area of its z-transform verifies $\{z \in \mathbb{C} : |z| > R_1\}$ (meaning also that $R_2 = +\infty$), and we have:*

$$x(0) = \lim_{z\to+\infty} X_z(z) \tag{14.12}$$

3. *The signal is anti-causal if and only if the convergence area of its z-transform verifies $\{z \in \mathbb{C} : |z| < R_2\}$ (meaning also that $R_1 = 0$).*

4. *If $X_z(z)$ is a rational function $B(z)/A(z)$, where the degrees of $B(z)$ and $A(z)$ are equal to Q and P respectively, the zeros are the Q roots of the polynomial $B(z)$, and the poles are the P roots of the polynomial $A(z)$.*

5. *If $X_z(z) = B(z)/A(z)$, the possible convergence areas are the non empty rings containing no poles and delimited by two poles. Mathematically speaking, a convergence area can be expressed:*

$$\{z \in \mathbb{C} : |p_j| < |z| < |p_k|\}$$

where p_j and p_k are two distinct roots of $A(z)$ defined so that $A(z) \neq 0$ for any z such that $|p_j| < |z| < |p_k|$. As a consequence, when using $B(z)/A(z)$, there are several possible converging areas, each one corresponding to a different sequence $\{x(n)\}$.

6. *If $X_z(z) = B(z)/A(z)$, a necessary condition for the sequence $\{x(n)\}$ to be causal is that the degree (with respect to z) of the numerator is less than, or equal to the degree (still with z as the variable) of the denominator. All we have to do then, in order to completely characterize the causal sequence, is to choose, as the convergence area:*

   $$\{z \in \mathbb{C} : |z| > |p_M|\}$$

   where p_M denotes the pole of $X_z(z)$ with the highest modulus. As a counter-example, you can check that no causal sequence corresponds to the function $X_z(z) = (z-1)^2/(z+1)$.

7. *If $X_z(z) = B(z)/A(z)$, and if the original corresponding sequence is real, then the roots of $A(z)$ and $B(z)$ are either real, or come in pairs of conjugate complex numbers.*

A5 Jury criterion

HINT: let $D(z) = A_n(z)$ be the transfer function's denominator such that the term with the highest degree has its coefficient equal to 1. We write:

$$A_n(z) = 1 + a_{n,1}z^{-1} + \cdots + a_{n,n}z^{-n}$$

where the $a_{n,k}$ coefficients are real.

- Such a polynomial is "unstable" if $|a_{n,n}| \geq 1$.
- In the case where $|a_{n,n}| < 1$, we cannot be sure that $A_n(z)$ is stable. By writing the polynomial $A_{n-1}(z)$:

  $$\begin{aligned} A_{n-1}(z) &= \frac{A_n(z) - a_{n,n}A_n^*(z)}{1 - a_{n,n}^2} \\ &= \frac{A_n(z) - a_{n,n}z^{-n}A_n(z^{-1})}{1 - a_{n,n}^2} \end{aligned} \tag{14.13}$$

  and we show that:

  $$A_n(z) \text{ stable} \Leftrightarrow A_{n-1}(z) \text{ stable}$$

– First, a few properties:

- the $A_{n-1}(z) \rightarrow A_n(z)$ relation:

  $$(1 - a_{n,n}^2)A_{n-1}(z) = A_n(z) - a_{n,n}z^{-n}A_n(z^{-1}) \tag{14.14}$$

$$\Rightarrow (1 - a_{n,n}^2)A_{n-1}(z^{-1}) = A_n(z^{-1}) - a_{n,n}z^n A_n(z)$$

$$\begin{aligned} \Rightarrow \quad & a_{n,n}z^{-n}(1 - a_{n,n}^2)A_{n-1}(z^{-1}) \\ = \quad & a_{n,n}z^{-n}A_n(z^{-1}) - a_{n,n}^2 A_n(z) \end{aligned} \tag{14.15}$$

[14.14] + [14.15] gives:

$$A_n(z) = A_{n-1}(z) + a_{n,n}z^{-n}A_{n-1}(z^{-1}) \tag{14.16}$$

- The roots vary continuously with $a_{n,n}$. Let z_0 be a root of z^n with a multiplicity of m of $A_n(z) = f(z)$, for a given value of $a_{n,n}$. $a_{n,n}$ undergoes a variation of δ. Can we find δ such that the m roots z_k of the new $z^n A_n(z) = g(z)$ remain in a neighborhood of z_0?

  According to [14.16], the variation of the function $z^n A_n(z)$ can be expressed:

  $$\Delta(z) = g(z) - f(z) = \delta A_{n-1}(z^{-1})$$

  Consider a disc with a contour γ surrounding z_0 in such a way that z_0 is the only root inside the disc. When z circles γ once counterclockwise, $f(z)$ travels counterclockwise around the origin m times. We can always choose δ so as to have $|\Delta(z)| = \delta|A_{n-1}(z^{-1})| < |f(z)|$. If f_0 is a lower-bound of $f(z)$ along γ, all we have to do is set:

  $$\delta < \frac{f_0}{\sup |A_{n-1}(z^{-1})|}$$

  Under these conditions, $g(z) = f(z) + \Delta(z)$ circles around the origin m times. This can be shown by writing:

  $$g(z) = f(z)\left(1 + \frac{\Delta(z)}{f(z)}\right)$$

  Because $|\Delta(z)| < |f(z)|$, the term $1 + \Delta(z)/f(z)$ cannot circle around the origin, leading us to the result (Rouché's theorem).

  The point of all this was to show that $g(z)$ has m roots inside γ, hence the continuity of the variations with $a_{n,n}$.

– Let us assume that $z^n A_{n-1}(z)$ is stable. Consider relation [14.16]. If a root of $z^n A_n(z)$ belongs to the unit circle, it can be written $e^{j\varphi}$ and:

$$a_{n,n} = -\frac{A_{n-1}(e^{j\varphi})}{e^{-jn\varphi}A_{n-1}(e^{-j\varphi})} \Rightarrow |a_{n,n}| = 1$$

If $a_{n,n} = 0$, the roots of $z^n A_n(z)$ are those of $z^n A_{n-1}(z)$ which are inside the unit circle (there are n of them with at least one at the origin). If $a_{n,n} = \pm\infty$ the roots are those of $z^{-1}A_{n-1}(z^{-1})$ (there are n of them with at least one at infinity). They are outside the unit disk. When $a_{n,n}$ varies, the roots all remain inside the unit disk so long as $|a_{n,n}| < 1$, which is true by hypothesis.

– Conversely, let us assume that $z^n A_n(z)$ is stable. We are going to prove that $z^n A_{n-1}(z)$ is stable. Notice that $a_{n,n}$, because it is the product of the roots, verifies $|a_{n,n}| < 1$. We will use proof by contradiction. We assume that $z^n A_{n-1}(z)$ has a root outside the unit disk. The method is the same as before. The roots of $z^n A_n(z)$ could not be outside the unit disk if $|a_{n,n}| > 1$, which contradicts our hypothesis.

Constructing the Jury table exactly corresponds to constructing the sequence of terms $A_n(z)$. The last condition applies to a second-degree polynomial the stability of which is ensured when $|a_{2,2}| < 1$. As soon as one of the $a_{k,k}$ becomes greater than 1, the system becomes unstable. ■

HINT: we can check that the first two conditions are met with proof by induction. If we are given $A_n(z) = 1 + a_1 z^{-1} + \cdots + a_n z^{-n}$, we have:

$$A_{n-1}(z) = 1 + \frac{a_1 - a_n a_{n-1}}{1 - a_n^2} z^{-1} + \cdots + \frac{a_{n-1} - a_n a_1}{1 - a_n^2} z^{-n+1}$$

If we assume that $A_{n-1}(1) > 0$, we can check that $A_n(1) > 0$, and vice versa. The same can be done for the conditions with $z = -1$. This means we don't have to take the calculation to its end (the calculation of $A_1(z)$). We can replace the last condition with the conditions on $A_n(1)$ and $A_n(-1)$, and not have to construct the array if one of these conditions is not met. ■

A6 FFT filtering algorithms revisited

We will now see another way of constructing FFT filtering algorithms. This presentation is based on a property of circulant matrices.

Definition 14.1 (Circulant matrix) *An $L \times L$ square matrix is said to be circulant if its l lines are obtained by circular translation of an L length vector.*

Thus, the matrix:

$$\mathbf{C} = \begin{bmatrix} c_1 & c_2 & c_3 & c_4 \\ c_4 & c_1 & c_2 & c_3 \\ c_3 & c_4 & c_1 & c_2 \\ c_2 & c_3 & c_4 & c_1 \end{bmatrix}$$

is a 4×4 circulant matrix.

Theorem 14.1 (DFT and circulant matrix) *Let* $\mathbf{C}$ *be an* $L \times L$ *circulant matrix. The eigendecomposition of* $\mathbf{C}$ *is:*

$$\mathbf{C} = \frac{1}{L}\mathbf{F}\mathbf{D}\mathbf{F}^H \tag{14.17}$$

where $\mathbf{F}$ *is the matrix known as the Fourier matrix with* $f_{kn} = e^{-2j\pi kn/L}$ *as its generating element, and* $\mathbf{D}$ *the diagonal matrix constructed from the DFT of the first line of* $\mathbf{C}$*, that is to say:*

$$d_{kk} = \sum_{n=0}^{L-1} c_n e^{-2j\pi kn/L} \tag{14.18}$$

All we have to do is multiply $\mathbf{C}$ on the right by the vector $\mathbf{e}_k = \begin{bmatrix}1 & e^{-2j\pi k/L} & \dots & e^{-2j\pi k(L-1)/L}\end{bmatrix}^T$, which is a column of $\mathbf{F}$. The result is $\mathbf{C}\mathbf{e}_k = d_{kk}\mathbf{e}_k$, and therefore $\mathbf{CF} = \mathbf{FD}$. Because the Fourier matrix verifies $\mathbf{F}\mathbf{F}^H = L\mathbf{I}_L$, we infer 14.17.

Example 14.1 (A fast filtering algorithm that uses the FFT)
Consider the filtering equation $y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$ (N length filter). Let $\mathbf{y} = [y(n), \dots, y(n-M+1)]^T$ be a column vector made up of M consecutive values of $y(n)$.

1. Show that $\mathbf{y} = \mathbf{Hx}$ where $\mathbf{H}$ is an $M \times (M+N-1)$ Toeplitz matrix constructed from $h(k)$ and where $\mathbf{x}$ is a column-vector made up of $(M+N-1)$ consecutive values of $x(n)$.

2. **Overlap-save**

   (a) By completing $\mathbf{H}$ with $N-1$ lines, show that:

   $$\begin{bmatrix}\mathbf{y}\\ \mathbf{z}\end{bmatrix} = \mathbf{Cx}$$

   where $\mathbf{C}$ is an $L \times L$ circulant matrix where $L = M+N-1$. From now on, L will be used to denote the DFT's length.

   (b) Use this result to find a calculation of $\mathbf{y}$ that uses the FFT algorithm.

3. **Overlap-add**
   The $L \times L$ Toeplitz matrix is denoted $\mathbf{K}$. It is constructed from the vector $\mathbf{h}^T$ with $(L-N)$ zeros, and it is an upper triangular matrix. Notice that the matrix $\mathbf{K}$ coincides with $\mathbf{C}$ (from the previous part), except in the bottom-left corner. $\mathbf{x}_p = [x(pM), \dots, x(pM-M+1)]^T$ is used to denote a block of $M = L-N+1$ consecutive input values. We assume:

   $$\mathbf{v}_p = \begin{bmatrix}\mathbf{0}_{N-1,1}\\ \mathbf{x}_p\end{bmatrix}$$

(a) Show that the $M = (L - N + 1)$ consecutive output values can be obtained with the expression:

$$\begin{bmatrix}\mathbf{I}_M & \mathbf{0}_{M,N-1}\end{bmatrix} \mathbf{K} \left(\begin{bmatrix}\mathbf{0}_{N-1,1} \\ \mathbf{x}_p\end{bmatrix} + \mathbf{J} \begin{bmatrix}\mathbf{0}_{N-1,1} \\ \mathbf{x}_{p+1}\end{bmatrix} \right)$$

where:

$$\mathbf{J} = \begin{bmatrix}\mathbf{0}_{N-1,M} & \mathbf{I}_{N-1} \\ \mathbf{0}_{M,M} & \mathbf{0}_{M,N-1}\end{bmatrix}$$

(b) Show that $\mathbf{K}$ and $\mathbf{J}$ commute with each other and that $\mathbf{K}\mathbf{v}_p = \mathbf{C}\mathbf{v}_p$.

(c) Use this result to determine a filtering algorithm that uses the FFT.

SOLUTION:

1. We have:

$$\mathbf{y} = \begin{bmatrix} h_0 & h_1 & \cdots & h_{N-1} & 0 & \cdots & 0 \\ 0 & h_0 & h_1 & \cdots & h_{N-1} & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \ddots & 0 \\ 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1} \end{bmatrix} \mathbf{x} = \mathbf{H}\mathbf{x}$$

$\mathbf{H}$ is a $M \times (M + N - 1)$ Toeplitz matrix.

2. **Overlap-save**

(a) By completing $\mathbf{H}$ with $(N - 1)$ lines with a length of $(M + N - 1)$, we get the length $L = (M + N - 1)$ circulant matrix:

$$\mathbf{C} = \begin{bmatrix} h_0 & h_1 & \cdots & h_{N-1} & 0 & \cdots & 0 \\ 0 & h_0 & h_1 & \cdots & h_{N-1} & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \ddots & 0 \\ 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1} \\ h_{N-1} & 0 & \cdots & 0 & h_0 & \cdots & h_{N-2} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ h_1 & \cdots & h_{N-1} & 0 & \cdots & 0 & h_0 \end{bmatrix}$$

By multiplying $\mathbf{x}$ on the right, we get:

$$\begin{bmatrix}\mathbf{y} \\ \mathbf{z}\end{bmatrix} = \mathbf{C}\mathbf{x}$$

where $\mathbf{z}$ is an "unwanted" length $(N - 1)$ vector.

(b) Using the previous equation and theorem 14.1 leads us to:

$$\mathbf{y} = \frac{1}{L} \begin{bmatrix} \mathbf{I}_M & \mathbf{0}_{N-1} \end{bmatrix} \mathbf{F}^H \mathbf{D} \mathbf{F} \mathbf{x}$$

where the diagonal matrix $\mathbf{D}$ has as its diagonal the DFT $\mathbf{h}_F$ of $\mathbf{h}$ calculated for $L = M + N - 1$ points. This leads us to an algorithm for calculating $\mathbf{y}$.

L is such that $L > N$, and usually is a power of 2. After having calculated once the L length DFT $\mathbf{h}_F$ of $\mathbf{h}$ (the diagonal of $\mathbf{D}$), the following operations are reiterated:

- calculation of the FFT of the L modulo $M = L - N + 1$ (overlap) length block $\mathbf{x}$ that leads to $\mathbf{x}_F = \mathbf{F}\mathbf{x}$;
- term-by-term multiplication of $\mathbf{x}_F$ by $\mathbf{h}_F$ that leads to $\mathbf{v}_F$;
- calculation of the inverse DFT of $\mathbf{v}_F$;
- the $M = L - N + 1$ first values of $\mathbf{v}$ are saved.

We end up with the same results as with the overlap-save algorithm.

3. **Overlap-add**

(a) The M first lines of the product of $\mathbf{K}$ by the vector $\mathbf{v} = \begin{bmatrix} x(pM + N - 1) & \ldots & x(pM + 1) & \mathbf{x}_p \end{bmatrix}^T$ are the filter's outputs at the times $pM + N - 1, \ldots$

$\mathbf{J}$ selects the last $(N - 1)$ lines of $\mathbf{x}_{p+1}$, that is $x(pM + N - 1)$, $\ldots$, $x(pM + 1)$. The linearity property allows us to demonstrate the expected result.

(b) $\mathbf{KJ} = \mathbf{JK}$ because of the fact that $\mathbf{K}$ is upper triangular and that the sub-matrices in the north-west and south-east corners coincide.

$\mathbf{K}\mathbf{v}_p = \mathbf{C}\mathbf{v}_p$ simply because $\mathbf{v}_p$ has $(N - 1)$ leading zeros.

(c) We can therefore write:

$$\begin{aligned} \mathbf{K}(\mathbf{v}_p + \mathbf{J}\mathbf{v}_{p+1}) &= \mathbf{C}\mathbf{v}_p + \mathbf{J}\mathbf{C}\mathbf{v}_{p+1} \\ &= \mathbf{F}^H \mathbf{D} \underbrace{\mathbf{F}\mathbf{v}_p}_{TFD} + \mathbf{J}\mathbf{F}^H \mathbf{D} \underbrace{\mathbf{F}\mathbf{v}_{p+1}}_{TFD} \end{aligned}$$

This leads to the algorithm: L is such that $L > 2N - 1$, and usually is a power of 2. After having calculated once the L length DFT $\mathbf{h}_F$ of $\mathbf{h}$ (the diagonal of $\mathbf{D}$), the following operations are reiterated:

- calculation of the FFT of the $M = L - N + 1$ length block $\mathbf{x}_{p+1}$, which is completed by $(N - 1)$ zeros (there is no overlap), leading to $\mathbf{x}_F = \mathbf{F}\mathbf{x}_{p+1}$;
- term-by-term multiplication of $\mathbf{x}_F$ by $\mathbf{h}_F$ which gives us $\mathbf{s}_{F,p+1}$;

- calculation of the inverse DFT of $\mathbf{s}_{F,p+1}$ which gives us $\mathbf{s}_{p+1}$;
- sum of the blocks $\mathbf{s}_{p+1}$ and $\mathbf{s}_p$ with an overlap of $(N-1)$ values.

We end up with the same results as with the overlap-add algorithm.

Bibliography

[1] B. S. Atal, V. Cuperman, and A. Gersho, editors. *Advances in Speech Coding.* Kluwer Academic Publishers, 1991.

[2] M. F. Barnsley and L. P. Hurd. *Fractal Image Compression.* AK Peters, Ltd., 1993.

[3] G. Battail. *Théorie de l'Information.* Collection Pédagogique de Télécommunication. Masson, 1997.

[4] M. Bellanger. *Traitement Numérique du Signal.* Collection CNET-ENST. Masson, 1980.

[5] A. Benveniste, M. Métivier, and P. Priouret. *Adaptive Algorithms and Stochastic Approximations.* Springer Verlag, 1990.

[6] P. Billingsley. *Probability and Measure.* J. Wiley, 1979.

[7] R.E. Blahut. *Fast Algorithms for Signal Processing.* Addison-Wesley, 1985.

[8] G. Blanchet and J. Prado. *Eléments d'Automatique.* Collection Pédagogique de Télécommunication. Ellipses, 1994.

[9] R. Boite and H. Leich. *Les Filtres Numériques.* Masson, Collection CNET-ENST, 1980.

[10] F. M. Boland, J. J. K. J. Ò Ruanaidh, and C. Dautzenberg. "Watermarking Digital Images for Copyright Protection". *Proceedings of the International Conference on Image Processing and its Applications*, July 1995. Edimburgh, Scotland.

[11] S.F. Boll. "Suppression of Acoustic Noise in Speech Using Spectral Substraction". *IEEE Trans. Acoust., Speech, Signal Processing,* pages 113–120, April 1979.

[12] P. Brémaud. *Introduction aux Probabilités.* Springer Verlag, 1988.

[13] P. Brémaud. *Signaux Aléatoires.* Collection de l'X, Ellipses, 1993.

[14] P. Brockwell and R. Davies. *Time Series: Theory and Methods.* Springer Verlag, 1990.

[15] J.M. Brossier. *Signal et Communication Numérique.* Hermès, 1997.

[16] J. P. Burg. "Maximum Entropy Spectral Analysis". PhD thesis, Standford Univ., 1975.

[17] C. S. Burrus, J.H. McClellan, A. V. Oppenheim, T.W. Parks, R. W. Schafer, and H. W. Schuessler. *Computer-Based Exercises for Signal Processing using Matlab.* Prentice Hall, 1994.

[18] J. Canny. "Computational Approach to Edge Detection". *IEEE Trans. Pattern Anal. Machine Intell.*, pages 679–698, November 1986.

[19] J. Capon. "High-resolution frequency-wavenumber spectrum analysis". *Proc. IEEE*, 57, no. 8:1408–1418, August 1969.

[20] H. Cartan. *Théorie Elémentaire des Fonctions Analytiques de une ou plusieurs Variables Complexes.* Hermann, Paris, 1975.

[21] M. Charbit. *Eléments de Théorie du Signal: Signaux Aléatoires.* Collection Pédagogique de Télécommunication. Ellipses, 1996.

[22] M. Charbit and G. Blanchet. "Eléments de Traitement Numérique du Signal". *Techniques de l'Ingénieur*, 1998.

[23] F. Le Chevalier. *Principes de Traitement des Signaux Radar et Sonar.* Masson, 1989.

[24] J.W. Cooley and J.W. Tuckey. "An Algorithm for the Machine Calculation of Complex Fourier Series". *Math. of Comp.*, 19:297–301, April 1965.

[25] R.E. Crochiere. "A Weighted Overlap-Add Method of Short-time Fourier Analysis/Synthesis". *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-28(2):99–102, February 1980.

[26] Ronald E. Crochiere and Lawrence R. Rabiner. *Multirate Digital Signal Processing.* Prentice Hall, Englewood Cliffs, 1983.

[27] J.P. Delmas. *Eléments de Théorie du Signal: Signaux Déterministes.* Collection Pédagogique de Télécommunication. Ellipses, 1995.

[28] J.P. Delmas. *Introduction aux Probabilités.* Ellipses, 2000.

[29] P. Duhamel. “Blind Equalization. Tutorial Conference”. *Proc. International Conference on Acoustics, Speech and Signal Processing*, 1995.

[30] P. Duvaut. *Traitement du Signal, Concepts et Applications.* Hermès, 1991.

[31] Al Bovik (editor). *Handbook of Image & Video Processing.* Academic Press, 2000.

[32] Y. Ephraim and D. Malah. “Speech Enhancement Using a Minimum Mean-square Error Short-time Spectral Amplitude Estimator”. *IEEE Trans. Acoust., Speech, Signal Processing*, pages 1109–1121, December 1984.

[33] Y. Ephraim and D. Malah. “Speech Enhancement Using a Minimum Mean-square Error Log-Spectral Amplitude Estimator”. *IEEE Trans. Acoust., Speech, Signal Processing*, pages 443–445, April 1985.

[34] Itakura F. “Minimum Prediction Residual Principle Applied to Speech Recognition”. *IEEE Transactions on Acoustics Speech and Signal Processing*, AS23.1:67–72, 1975.

[35] J. L. Flanagan. *Speech Analysis, Synthesis, and Perception.* Springer Verlag, New York, 1972.

[36] J. L. Flanagan and R. M. Golden. “Phase Vocoder”. *Bell System Technical Journal*, pages 1493–1509, November 1966.

[37] P. Flandrin. *Temps-Fréquence.* Hermès, 1993.

[38] B. Friedlander. “Lattice Methods for Spectral Estimation”. *Proc. IEEE*, 70, 1982.

[39] W. Gardner. *Cyclostationarity in Communications and Signal Processing.* IEEE Press, 1994.

[40] A. Gersho and R.M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, Boston, 1992.

[41] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The John Hopkins University Press, 1989.

[42] J. P. Guillois. *Techniques de Compression des Images.* Hermès, 1996.

[43] F.J. Harris. “On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform”. *Proc. IEEE*, 66:51–83, January 1978.

[44] S. Haykin. *Adaptive Filter Theory.* Prentice Hall, Englewood Cliffs, NJ, USA, 2nd edition, 1991.

[45] S. Haykin. *Communication Systems.* John Wiley & Sons, New York, NY, 4th edition, 2001.

[46] P. Hough. “Method for recognizing complex patterns”, 1962. US Patent 3069654.

[47] M.R. Iseli and A. Alwan. “Inter- and Intra-speaker Variability of Glottal Flow Derivative using the LF Model”. *6th International Conference on Spoken Language Processing,* pages 477–480, 2000.

[48] L. B. Jackson. *Digital Filters and Signal Processing with Matlab Exercises.* Kluwer Academic Publishers, 3rd edition, 1995.

[49] N. S. Jayant and P.Noll. *Digital Coding of Waveforms: Principles and Applications to Speech and Video.* Prentice Hall Signal Processing Series, 1984.

[50] M. Joindot and A. Glavieux. *Introduction aux Communications Numériques.* Collection Pédagogique de Télécommunication. Ellipses, 1995.

[51] T. Kailath. *Linear Systems.* Prentice Hall, Englewood Cliffs, 1980.

[52] S. M. Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory.* Prentice Hall, 1993.

[53] S. M. Kay and S. L. Marple JR. “Spectrum Analysis – A Modern Perspective”. *Proc. IEEE,* 69(11):1380–1418, November 1981.

[54] S.M. Kay. *Modern Spectral Estimation, Theory and Application.* Prentice Hall, Englewood Cliffs, 1988.

[55] R. Kumaresan and D. W. Tufts. “Estimating the Parameters of Exponentially Damped Sinusoids and Pole–Zero Modeling in Noise”. *IEEE Trans. Acoust., Speech, Signal Processing,* ASSP-30(6):833–840, December 1982.

[56] M. Kunt. *Traitement Numérique des Signaux.* Presses polytechniques romandes, 1980.

[57] J. Laroche. “The Use of the Matrix Pencil Method for the Spectrum Analysis of Musical Signals”. *JASA,* 94(4):1958–1965, October 1993.

[58] Y. Linde, A. Buzzo, and R. Gray. “An Algorithm for Vector Quantizer Design”. *IEEE Trans. on Communications,* COM-28:84–95, January 1980.

[59] S.P. Lloyd. “Least Squares Quantization in PCM”. *IEEE Trans. on Information Theory,* pages 129–137, March 1982.

[60] O. Macchi. *Adaptive Processing: The Least Mean Squares Approach with Applications in Transmission.* John Wiley & Sons, Inc., 1995.

[61] S. Mallat. *A Wavelet Tour of Signal Processing.* Academic Press, London, 1999.

[62] S. Mallat and F. Falzon. "Analysis of Low Bit Rate Image Transform Coding". *IEEE Trans. Signal Processing*, 46:1027–1042, April 1998.

[63] H. S. Malvar. *Signal Processing with Lapped Transforms.* Artech House, 1992.

[64] J. Max. "Quantizing for Minimum Distorsion". *IRE Trans. on Information Theory*, pages 7–12, March 1960.

[65] F. Mintzer, A. Cazes, F. Giordano, J. Lee, K. Magerlein, and F. Schiattarella. "Capturing and Preparing Images of Vatican Library Manuscripts for Access via Internet". *Proceedings of the International Conference on Imaging Science, Systems and Technology, Las Vegas*, 43, June 1997.

[66] N. Moreau. *Techniques de Compression des Signaux.* Masson, 1995.

[67] E. Moulines and F. Charpentier. "Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones". *Speech Communication*, 5-6(9):453–467, December 1990.

[68] E. Moulines, P. Duhamel, J.F. Cardoso, and S. Mayrargue. "Subspace Methods for the Blind Identification of Multichannel FIR Filters". *IEEE Trans. on Signal Processing*, 43(2):516–525, February 1995.

[69] A. Oppenheimer and R. Shafer. *Discrete-Time Signal Processing.* Prentice Hall, 1989.

[70] N. Otsu. "A Threshold Selection Method from Gray-Level Histograms". *IEEE Trans. on Syst. Man and Cyber.*, 1:62–69, 1979.

[71] A. Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill, New York, 3rd edition, 1991.

[72] B. Picinbono. *Théorie des Signaux et Systèmes.* Dunod, 1988.

[73] B. Porat. *Digital Processing of Random Signals : Theory and Methods.* Prentice Hall, Englewood Cliffs, 1994.

[74] B. Porat. *A Course in Digital Signal Processing.* John Wiley & Sons, Inc., 1997.

[75] B. Porat and B. Friedlander. "On the Accuracy of the Kumaresan-Tufts Method for Estimating Complex Damped Exponentials". *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-35:232–235, February 1987.

[76] M.B. Priestley. *Spectral Analysis and Time Series, vol.1.* Academic Press, London, 1981.

[77] J.G. Proakis. *Digital Communications.* McGraw-Hill, 4th edition, 2000.

[78] J.G. Proakis and D.M. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1996.

[79] T. Kailath R. Roy. "ESPRIT-estimation of signal parameters via rotational invariance techniques". *IEEE Trans. on Acoust. Speech, Signal Processing*, ASSP-37:984–995, August 1989.

[80] D.C. Rife and R.R. Boorstyn. "Multiple-Tone Parameter Estimation from Discrete-Time Observations". *Bell Systems Tech. J.*, 55(9):1389–1410, November 1976.

[81] O. Rioul and P. Duhamel. "Fast Algorithms for Wavelet Transform Computation". *Wavelets in Biomedical Signal Processing*, 1997.

[82] M. Rosenblatt. *Stationary Processes and Random Fields.* Birkhauser, 1985.

[83] R. Roy, A. Paulraj, and T. Kailath. "Esprit: A Subspace Rotation Approach to Estimation of Parameters of Cisoids in Noise". *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-34:1340–1342, October 1986.

[84] L. L. Scharf. *Statistical Signal Processing: Detection, Estimation, and Time Series Analysis.* Addison Wesley, 1991.

[85] A. Schuster. "On the Periodicities of Sunspots". *Philosophical Transactions of the Royal Society of London*, 206, Ser.A:60–100, April 1906.

[86] R. J. Serfling. *Approximation theorems of mathematical statistics.* Wiley series in probability and mathematical statistics. John Wiley & sons, 1980.

[87] C. Shannon. "A Mathematical Theory of Communication". *Bell Systems and Technics Journal*, vol.: 19:379–423 (part I), 623–656 (part II), 1948.

[88] C. Shannon. "Coding Theorems for a Discrete Source with a Fidelity Criterion". *IRE National Convention Record*, pages 142–163, Part 4, 1959.

[89] A. Skorokhod and I. Guikhman. *Introduction à la Théorie des Processus Aléatoires.* Presses Polytechniques Romandes, 1980.

[90] P. Stoica and A. Nehorai. “MUSIC, Maximum Likelihood, and Cramér-Rao Bound: Further Results and Comparisons”. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-38(12):2140–2150, December 1990.

[91] C.W. Therrien. *Discrete Random Signals and Statistical Signal Processing.* Prentice Hall signal processing series, Englewood Cliffs, NJ, USA, 1992.

[92] Cover T.M. and Thomas J.A. *Elements of Information Theory.* John Wiley, New York, 1991.

[93] Union Internationale des Télécommunications, CCITT. “Technologie de l'information - Compression numérique et codage des images fixes - de nature photographique - Prescriptions et lignes directrices”, 1981.

[94] P. P. Vaidyanathan. “Quadrature Mirror Filter Banks, M-band Extensions and Perfect-Reconstruction Techniques”. *IEEE ASSP Mag.*, 4(3):4–20, 1987.

[95] P. P. Vaidyanathan. “Multirate Digital Filters, Filter Banks, Polyphase Networks, and Applications: A Tutorial”. *Proc. IEEE*, 78(1):56–93, January 1990.

[96] P. P. Vaidyanathan. *Multirate Systems and Filter Banks.* Prentice Hall, Englewood Cliffs, 1993.

[97] D. Ventre. *Communications Analogiques.* Collection Pédagogique de Télécommunication. Ellipses, 1991.

[98] M. Vetterli and J. Kovčević. *Wavelets and Subband Coding.* Prentice Hall, Englewood Cliffs, 1995.

[99] G. K. Wallace. “The JPEG Still Picture Compression Standard”. *Communications of the ACM*, April 1991.

[100] E. Wang. *Stochastic Processes in Information and Dynamical Systems.* Mac Graw Hill, 1971.

[101] P. D. Welch. “The Use of Fast Fourier Transforms for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short Modified Periodograms”. *IEEE Trans. Audio Electroacoust.*, AU-15, June 1967.

[102] B. Widrow and S. Stearns. *Adaptive Signal Processing.* Prentice Hall, 1985.

[103] R. B. Wolfgang and E. J. Delp. “Overview of Image Security Techniques with Applications in Multimedia Systems”. *Proceedings of the SPIE Conference on Multimedia Networks: Security, Displays, Terminals, and Gateways*, 3228:297–308, November 1997.

[104] G. Yang, H. Leich, and R. Boite. “Multiband Code-Excited Linear Prediction (MBCELP) for speech coding”. *Signal Processing*, 31(2):215–228, March 1993.

[105] G. U. Yule. “On a Method of Investigating Periodicities in Disturbed Series, with special reference to Wolfer’s Sunspot Numbers”. *Philosophical Transactions of the Royal Society of London*, 226, Ser.A:267–298, April 1927.

Index